HZ BOOKS
华章教育

Jones and Bartlett

计 算 机 科 学 丛 书

原书第

计算机科学概论

（美） Nell Dale 得克萨斯大学奥斯汀分校 John Lewis 维拉诺瓦大学 著 张欣 胡伟 等译



Computer Science Illuminated

Third Edition

机械工业出版社
China Machine Press

VISIT…

LANZAROTE Caliente.com

计算机科学概论（原书第3版）

本书由当今该领域备受赞誉且经验丰富的教育家Nell Dale和John Lewis共同编写，全面介绍计算机科学领域的基础知识，为广大学生勾勒了一幅生动的画卷。就整体而言，全书内容翔实、覆盖面广，旨在向读者展示计算机科学的全貌；从细节上看，本书层次清晰、描述生动，基于计算机系统的洋葱式结构，分别介绍信息层、硬件层、程序设计层、操作系统层、应用程序层和通信层，涉及计算机科学的各个层面。

本书贯穿了计算机科学系统的各个方面，非常适合作为计
业课程打下坚实的基础；同时还适合作为非计算机专业的计算
的介绍。

本书第3版经过全面的修订和更新，具有以下特色：

- 采用一般语言对编程概念进行了描述。
- 彻底更新了名人传记、历史点评以及技术发展的前
的影响进行了探讨。
- 新增了图形学、信息安全、计算机安全、密码学和电子商务方面的介绍。
- 更新了计算机科学的最新发展现状的内容。
- 包含大量练习和思考题，方便教学。

本书的配套网站（http://csilluminated.jbpub.com/3e/）集学术性及趣味性于一体，提供了大量教学资源：Flash幻灯片、在线术语表、互动学习、填字游戏、HTML帮助信息等，以更灵活的方式多方位地加深读者对本书知识的理解。

作者简介



Nell Dale 计算机科学领域广受推崇的教育家。她于得克萨斯大学奥斯汀分校获得了数学硕士学位以及计算机科学博士学位。在得克萨斯大学奥斯汀分校执教25年，由于对计算机科学教学领域的突出贡献，1996年她获得了ACM SIGCSE计算机科学教育最杰出贡献奖。



John Lewis 计算机科学领域著名的教育家和作家，目前为维拉诺瓦大学计算机科学系副教授。他编写的Java软件及程序设计教材在该类书籍的排行榜上位居前列。他获得了很多教学方面的奖，包括大学优秀教学奖和杰出教学Goff奖。他的研究兴趣包括面向对象技术、多媒体以及软件工程。



影印版
书号：978-7-111-23516-3
定价：66.00元

上架指导：计算机/计算机科学概论
ISBN 978-7-111-17016-7
9 787111 170167

投稿热线：(010) 88379604
购书热线：(010) 68995259, 68995264
读者信箱：hzjsj@hzbook.com

华章网站 http://www.hzbook.com
网上购书：www.china-pub.com

封面设计·锡彬

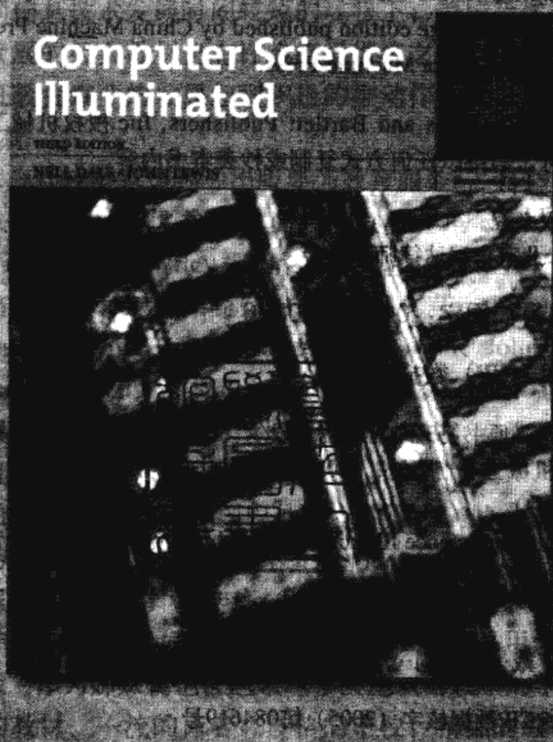
定价：49.00元

计 算 机 科 学 丛 书

原书第3版

计算机科学概论

(美) Nell Dale 得克萨斯大学奥斯汀分校 John Lewis 维拉诺瓦大学 著 张欣 胡伟 等译



Computer Science Illuminated

Third Edition

机械工业出版社
China Machine Press

本书由两位知名的计算机科学教育家编写，全面而细致地介绍了计算机科学的各个方面。书中从信息层开始，历经硬件层、程序设计层、操作系统层、应用程序层和通信层，深入剖析了计算系统的每个分层，最后讨论了计算的限制。此外，正文中穿插了大量的名人传记、历史点评、道德问题和最新的技术发展信息，有助于你进一步了解计算机科学。每章后面都附带有大量的练习，可以帮助你即时重温并掌握这一章所讲述的内容。

本书是计算机科学引论课程的理想教材，对于想要了解计算机科学概况的非专业人员，本书也是一个很好的选择。

Nell Dale and John Lewis: Computer Science Illuminated, Third Edition (ISBN 978-0-7637-4149-5).

Copyright © 2007 by Jones and Bartlett Publishers, Inc.

Original English language edition published by Jones and Bartlett Publishers, Inc., 40 Tall Pine Drive, Sudbury, MA 01776.

All rights reserved. No change may be made in the book including, without limitation, the text, solutions, and the title of the book without first obtaining the written consent of Jones and Bartlett Publishers, Inc. All proposals for such changes must be submitted to Jones and Bartlett Publishers, Inc. in English for his written approval.

Chinese simplified language edition published by China Machine Press.

Copyright © 2009 by China Machine Press.

本书中文简体字版由Jones and Bartlett Publishers, Inc.授权机械工业出版社独家出版。未经出版者书面许可，不得以任何方式复制或抄袭本书内容。

版权所有，侵权必究。

本书法律顾问　北京市展达律师事务所

本书版权登记号：图字：01-2008-0326

图书在版编目（CIP）数据

计算机科学概论（原书第3版）/（美）戴尔（Dale, N.），（美）刘易斯（Lewis, J.）著；张欣等译. —北京：机械工业出版社，2009.2

（计算机科学丛书）

书名原文：Computer Science Illuminated, Third Edition

ISBN 978-7-111-17016-7

Ⅰ.计…　Ⅱ.①戴…　②刘…　③张…　Ⅲ.计算机科学　Ⅳ.TP3

中国版本图书馆CIP数据核字（2005）第084619号

机械工业出版社（北京市西城区百万庄大街22号　邮政编码　100037）

责任编辑：迟振春

北京牛山世兴印刷厂印刷

2009年2月第1版第1次印刷

184mm × 260mm · 24.5印张

标准书号：ISBN 978-7-111-17016-7

定价：49.00元

凡购本书，如有倒页、脱页、缺页，由本社发行部调换

本社购书热线：(010) 68326294

出版者的话

文艺复兴以降，源远流长的科学精神和逐步形成的学术规范，使西方国家在自然科学的各个领域取得了垄断性的优势；也正是这样的传统，使美国在信息技术发展的六十多年间名家辈出、独领风骚。在商业化的进程中，美国的产业界与教育界越来越紧密地结合，计算机学科中的许多泰山北斗同时身处科研和教学的最前线，由此而产生的经典科学著作，不仅擘划了研究的范畴，还揭示了学术的源变，既遵循学术规范，又自有学者个性，其价值并不会因年月的流逝而减退。

近年，在全球信息化大潮的推动下，我国的计算机产业发展迅猛，对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇，也是挑战；而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短的现状下，美国等发达国家在其计算机科学发展的几十年间积淀和发展的经典教材仍有许多值得借鉴之处。因此，引进一批国外优秀计算机教材将对我国计算机教育事业的发展起到积极的推动作用，也是与世界接轨、建设真正的世界一流大学的必由之路。

机械工业出版社华章分社较早意识到“出版要为教育服务”。自1998年开始，华章分社就将工作重点放在了遴选、移译国外优秀教材上。经过多年的不懈努力，我们与Pearson，McGraw-Hill，Elsevier，MIT，John Wiley & Sons，Cengage等世界著名出版公司建立了良好的合作关系，从他们现有的数百种教材中甄选出Andrew S. Tanenbaum，Bjarne Stroustrup，Brain W. Kernighan，Dennis Ritchie，Jim Gray，Afred V. Aho，John E. Hopcroft，Jeffrey D. Ullman，Abraham Silberschatz，William Stallings，Donald E. Knuth，John L. Hennessy，Larry L. Peterson等大师名家的一批经典作品，以“计算机科学丛书”为总称出版，供读者学习、研究及珍藏。大理石纹理的封面，也正体现了这套丛书的品位和格调。

“计算机科学丛书”的出版工作得到了国内外学者的鼎力襄助，国内的专家不仅提供了中肯的选题指导，还不辞劳苦地担任了翻译和审校的工作；而原书的作者也相当关注其作品在中国的传播，有的还专程为其书的中译本作序。迄今，“计算机科学丛书”已经出版了近两百个品种，这些书籍在读者中树立了良好的口碑，并被许多高校采用为正式教材和参考书籍。其影印版“经典原版书库”作为姊妹篇也被越来越多实施双语教学的学校所采用。

权威的作者、经典的教材、一流的译者、严格的审校、精细的编辑，这些因素使我们的图书有了质量的保证。随着计算机科学与技术专业学科建设的不断完善和教材改革的逐渐深化，教育界对国外计算机教材的需求和应用都将步入一个新的阶段，我们的目标是尽善尽美，而反馈的意见正是我们达到这一终极目标的重要帮助。华章分社欢迎老师和读者对我们的工作提出建议或给予指正，我们的联系方法如下：

华章网站：www.hzbook.com

电子邮件：hzjsj@hzbook.com

联系电话：(010) 88379604

联系地址：北京市西城区百万庄南街1号

邮政编码：100037

HZ BOOKS 华章教育

译者序

《计算机科学概论》是每一个学习计算机科学的人都应该读的第一本书。当我翻译完整本书后，一直在想，为什么在我的大学课程中，没有这样一门课，能够系统地讲述一遍计算机科学发展的历史，让我在学习的一开始就对计算机科学有一个整体的认识。这本书除了系统地介绍整个计算机系统外，还讲述了计算机系统的发展史。在阅读每个章节时，你都会明白这个章节要介绍的硬件、软件、语言等的来龙去脉，这样就能为以后的深入研究打下坚实的基础。

本书的主旨就是给初学者提供一本全面了解计算机科学的教材。本书的作者具有丰富的实际教学经验，真正了解初学者需要什么，并且集思广益，使得本书的内容更加完善。在这本书中，作者用了一个形象的比喻，把计算机系统比作洋葱，它们的相似之处就在于内部结构都是一层层的。

第1章是基础篇，介绍了硬件和软件的历史，以及计算机系统的洋葱式结构。以后的各章就根据这种结构，分别介绍了计算机系统的信息层、硬件层、程序设计层、操作系统层、应用程序层和通信层，最后则总结性地讨论了计算机硬件和软件固有的局限性，以及计算机能够解决和不能解决的问题。

除了详细地介绍计算机系统的方方面面外，本书还有三个亮点。第一，在每一章中都有一篇名人传记，记述了对计算机科学的发展做出过杰出贡献的人的生平。你可以想到的计算机界的传奇人物，几乎都可以在本书中找到他或她的踪影。第二，在每一章的结尾，附有一篇涉及法律和道德的短文，探讨了计算机科学发展史上出现过的一些有争议的问题。通过这些短文，你一定会对计算机科学有更进一步的了解。第三，每章后面都附带有大量的练习，可以帮助你即时重温这一章所讲述的内容，有助于你更好地掌握这些内容。

本书很适合作为计算机科学专业学生的入门教材。不过，即使是非计算机专业的学生和非专业人员，想要了解计算机科学的概况，本书也不失为一个很好的选择。

本书由张欣组织翻译和审校，参与翻译的还有胡伟、何健辉、黄璜、白佳、卞雨桂、陈洁、成洁、杜鲲、李才应、刘天成、刘吟、明卫军、潘秀燕、钱金蕾、王华红、魏胜、阎哲、王林、陈思锦、金川。

在翻译过程中，我们力求忠实、准确地把握原著的内容，但由于译者水平有限，书中难免有错误和不准确之处，敬请广大读者批评指正。

张 欣

2008年6月

前　言

论题选择

为了制定这本CS0教材的论题大纲，我们利用了许多资源，包括课程目录、教材大纲以及一个电子邮件调查问卷。设计这个调查问卷的目的在于了解我们的同事对这门课应该包括哪些内容的想法。我要求大家（包括自己）列出下列三种清单：

- 如果CS0这门课是学生在大学阶段学习的唯一一门计算机科学的课程，请列出四种你认为他们应该掌握的论题。
- 请列出四种你想要学生在进入CS1这门课之前掌握的论题。
- 请补充四种你想要学习CS1的学生熟悉的论题。

这些资源的交集反映出的大多数人的意见构成了本书的大纲。在学习CS1之前掌握了本书内容的学生将为继续学习计算机科学打下坚实的基础。尽管我们的意图是编写一本CS0教材，但是许多评论家都认为本书的覆盖面非常广泛，可以作为一种程序设计语言的计算机科学导论的参考书。

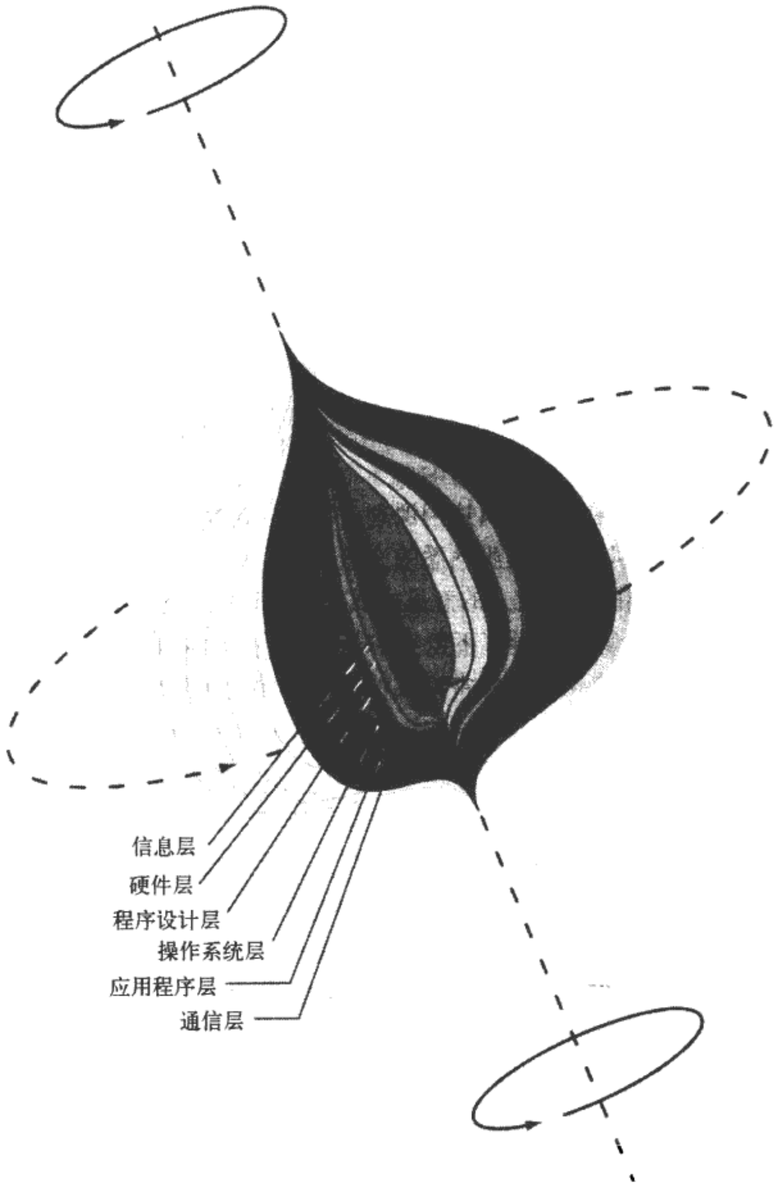
结构说明

在第1章中，我们介绍了硬件和软件的历史，并且用洋葱的结构来类比计算机系统的结构。计算机和它的机器语言构成了洋葱的芯，软件层和更复杂的硬件一层层地裹住了这个芯。首先介绍的是机器语言，然后是高级语言，包括FORTRAN、Lisp、Pascal、C、C++和Java。在介绍这些语言的同时，还介绍了利用它们进行程序设计的过程，包括自顶向下的设计和面向对象的设计。我们对抽象数据类型及其实现所扮演的角色的理解已经成熟了。操作系统和它的资源管理技术（包括更大更快的二级存储介质上的文件）包围着这些程序，并对它们进行管理。

接下来的一层由更复杂的通用或专用软件系统构成，它们覆盖了操作系统。这些功能强大的程序由计算机科学中的并行理论支持。最后一层由网络和网络软件构成，网络软件包括计算机之间通信必需的所有工具。Internet和万维网给这一层画上了最后一笔。

当这些层随着时间的推移逐渐出现时，用户对计算机系统的硬件接触得越来越少。每个层都是它下面的计算机系统的抽象。随着每个层的发展，新层的用户和内部层的用户联合起来，在经济领域的高科技部门创造了大量的生产力。本书的目的是提供各个层的概述，介绍基本的硬件和软件技术，使学生了解和欣赏计算系统的方方面面。

在介绍这种洋葱式结构时，我们有两种选择，一种是从内向外逐层介绍，另一种是从外向内进行介绍。从外向内的方法看起来非常吸引人。我们可以从最抽象的层开始介绍，一次剥掉一个层，直到具体的机器层为止。但是，研究表明，比起抽象的例子，学生们更容易理解具体的例子，即使他们本身是抽象思想家。因此，我们选择从具体的机器层开始，按照层的创建顺序进行分析，当学生完全理解了一个层之后，再转移到下一个层就比较容易。



第3版中的变化

一本新书的早期版本就像一艘新船的试航。如果设计得足够好，那么就只有一些小问题需要调整。当这本书过时了，那么它的新版本可能会要求全面修订。在计划这次改版时，我们咨询了CS教育学院的同事，要求他们就下面的问题给予一些反馈：我们应该进行哪些改动？哪些修订是必需的？50多位读者一致认为要在第3版中进行一些更新，但不是全面修订。因此，我们更新了一些名人传记，把“旧”的轶闻趣事换成了新的，重写了“道德问题”一节，使其反映当前的问题。在内容安排上，添加了有关图形学、信息安全、计算机安全、密码学以及电子商务方面的知识，我们将会在下面的“摘要”中讨论内容的具体变化。

有些人要求我们不把练习答案放在本书后面。与前几版一样，所有练习的答案都放在网上，教师可根据需要索取访问口令。

摘要

第1章是我们探索计算机科学（计算机系统“洋葱”）的基础，描述了本书的组织结构。第2章和第3章则分析了包含在物理硬件中的层。这个层称为*信息层*，它反映了如何在计算机上表示信息。第2章介绍了二进制数制以及它与其他数制（如人们日常用的十进制系统）的关系。第3章研究了如何获取多种类型（如数字、文本、图像、音频和视频）的信息以及如何用二进制格式表示它们。此外，本章还简短讨论了数据、信息和PNG图像格式的区别。

第4章和第5章介绍了*硬件层*。计算机硬件包括的设备有晶体管、门和电路，它们都按照基本原理控制电流。正是这些核心电路，使专用的元件（如计算机的中央处理器——CPU）得以运转。第4章介绍了门和电路。第5章介绍了计算机的元件，以及在冯·诺伊曼体系结构中这些元件是如何交互的。当然，第5章开头的广告已经更新了，在你阅读本书时，它可能又过时了。

第6～9章介绍了*程序设计层*。第6章分析了问题求解过程，同时涉及人类和计算机的问题求解方法。引导这个论题的是George Polya的人类问题求解策略。第6章的改动很大。该章引入了用伪代码编写算法的功能。自顶向下的设计和面向对象的设计都有相关的示例，同时有详细的伪代码说明。第7章使用模拟计算机Pep/7介绍了机器语言和汇编语言的概念，并将简单的伪代码算法转化成机器代码和汇编语言。此外还加入了一个循环示例。第8章介绍了高级程序设计语言的概念。我们用四种程序设计语言设计了一些小例子来说明这些伪代码概念，它们是Ada、VB.NET、C++和Java。第9章强调了抽象数据类型和数据结构在程序设计过程中的作用。

第10章和第11章介绍了*操作系统层*。第10章讨论了操作系统的资源管理任务，介绍了一些用于实现这些任务的基本算法。第11章介绍了文件系统，包括什么是文件系统，以及操作系统如何管理它们。第10章增加了关于设备驱动程序的讨论。

第12～14章介绍了*应用程序层*。这一层由人们用来解决问题的通用应用程序和专用应用程序构成。我们根据这些程序的基础，把这一层分到了计算机科学的几个子学科中。第12章分析了信息系统，第13章分析了人工智能，第14章分析了模拟、图形学和其他应用。我们还在第12章中加入了一个重要的新小节——信息安全，讨论数据的机密性、完整性和可用性，此外还加入一小节来讨论密码学。第14章中还纳入了关于图形学、电子商务和计算机安全的内容。

第15章和第16章介绍了*通信层*。第15章说明了计算机之间通信的理论和应用。第16章介绍了万维网和它对当今生活的影响。第16章增加了关于博客的小节。

从第2～16章都是说明计算机可以做什么以及它是如何做的。第17章进行了总结，讨论了计算机硬件和软件的固有局限性，以及计算机能够解决和不能解决的问题。在讨论算法的有效性时，采用了大O符号，以便讨论算法的分类。此外还介绍了停机问题，以说明某些不能解决的问题。

本书的第1章和最后一章就像一对书挡，第1章说明了计算系统是什么，第17章告诉我们计算系统不能做什么。它们之间的章节则深入探讨了构成计算系统的各个层。

为什么不介绍一种语言

本书原来的大纲中包括“Java入门”一章。有些评论家对本书是否应该包括对一种语言的介绍感到很矛盾，而另一些评论家则质疑为什么包括Java语言而不是C++。最终我们决定让用户自己选择，在本书的网站上有Java语言入门、C++语言入门、Visual Basic.NET语言入门、Python语言入门、Alice语言入门和Pascal语言入门的章节，它们的格式与本书的设计一致。

如果除了本书中的背景资料，学生还有能力掌握一种语言的基本语法和语义，那么可以联系本书的原出版社或访问本书原英文书网站http://csilluminated.jbpub.com/。另外，还可以将这些文章作为具有较强背景的学生的补充资料。

特性

第3版具有三种特性，用于强调计算的历史、广度以及新技术带来的道德义务。首先，每一章都有一个简短的名人传记，介绍对计算做出杰出贡献的人。这些人包括对数据层做出贡献的George Boole和Ada Lovelace，以及对通信层做出贡献的Doug Engelbart和Tim Berners-Lee。这些传记的目的是让学生了解计算界的历史以及那些对计算界做过贡献和正在做贡献的人。

我们称第二个特性为“标注”，因为没有更好的词可以表达它。标注显示在方框中，是过去、现在和未来的一些轶闻趣事，它们来自历史记录、当今的报纸和作者的见闻。这些小插曲的目的是使学生开心、鼓舞他们、激发他们的兴趣，当然还为了教育他们。

第三个特性是每章中的“道德问题”小节。这些小节的目的是说明在利用计算的好处时要承担的义务。隐私权、黑客、病毒和言论自由都属于我们的论题。在每章练习的结尾处有一个思考题部分，涉及这些道德问题和这一章的内容。

本书的网站

本书的网站（http://csilluminated.jbpub.com）包括大量的补充信息。在这个网站上有更多的名人传记、更多的标注以及关于道德问题的更新。此外，该站点还提供各种练习的电子学习工具，例如，纵横拼字谜和学生数字实验室。

该站点为教师提供了所有练习的答案、每一章的PowerPoint讲稿。

此外，网站上还有其他的练习，它们使用了新的格式。

致谢

对于这一版来说，读者是最有用的信息和建议来源。衷心感谢53位受调查者花时间填写了我们的网络调查问卷。还要感谢第1版和第2版的审校者，以及这一版的审校者，他们是：

Tim Bower，堪萨斯州立大学；Mikhail Brikman，Salem州立学院；Jacques Carette，McMaster大学；Howard Francis，Pikeville学院；Jim Jones，Graceland大学；Murray Levy，西洛杉矶学院；Lew Lowther，York大学；Jeffrey McConnell，Canisius学院；Richard Schlesinger，Kennesaw州立大学；Richard Spinello，Boston学院；Herman Tavani，Rivier学院；Amy Woszczynski，Kennesaw州立大学。

特别感谢Canisius学院的Jeffrey McConnell，他编写了第14章中的图形学部分；感谢

Rivier学院的Herman Tavani，他参与了修改“道德问题”的工作；感谢Boston学院的Richard Spinello，他撰写了博客的道德问题一文。还要感谢Luther学院的Bradley Miller和David Ranum，他们撰写了重要的一章；感谢Kennesaw州立大学的Richard Schlesinger，他提供了VB.NET入门的章节；感谢Kennesaw州立大学的Jose Garrido，他撰写了Alice程序设计入门的章节。

我还必须感谢我的网球朋友使我有一个健康的体魄，感谢我的桥牌朋友使我的头脑十分机敏，感谢我的家人做我的坚强后盾。

目　录

出版者的话
译者序
前言

第一部分　基　础　篇

第1章　全景图 ……1
1.1　计算系统 ……1
1.1.1　计算系统的分层 ……2
1.1.2　抽象 ……3
1.2　计算的历史 ……5
1.2.1　计算硬件的简史 ……5
1.2.2　计算软件的简史 ……11
1.2.3　预言 ……15
1.3　计算工具和计算学科 ……15
小结 ……16
道德问题：数字化分裂 ……17
练习 ……17
思考题 ……18

第二部分　信　息　层

第2章　二进制数值和记数系统 ……21
2.1　数字和计算 ……21
2.2　位置记数法 ……22
2.2.1　二进制、八进制和十六进制 ……25
2.2.2　其他记数系统中的运算 ……26
2.2.3　以2的幂为基数的记数系统 ……26
2.2.4　把十进制数转换成其他数制的数 ……28
2.2.5　二进制数值和计算机 ……29
小结 ……30
道德问题：计算机和国家安全 ……30
练习 ……31
思考题 ……32
第3章　数据表示法 ……34
3.1　数据和计算机 ……34
3.1.1　模拟数据和数字数据 ……35
3.1.2　二进制表示法 ……36
3.2　数字数据的表示法 ……38
3.2.1　负数表示法 ……38
3.2.2　实数表示法 ……40
3.3　文本表示法 ……42
3.3.1　ASCII字符集 ……43
3.3.2　Unicode字符集 ……44
3.3.3　文本压缩 ……44
3.4　音频信息表示法 ……47
3.4.1　音频格式 ……49
3.4.2　MP3音频格式 ……49
3.5　图像和图形的表示法 ……49
3.5.1　颜色表示法 ……49
3.5.2　数字化图像和图形 ……50
3.5.3　图形的矢量表示法 ……52
3.6　视频表示法 ……53
小结 ……53
道德问题：MGM Studios公司和Grokster有限公司 ……54
练习 ……55
思考题 ……57

第三部分　硬　件　层

第4章　门和电路 ……59
4.1　计算机和电学 ……59
4.2　门 ……61
4.2.1　非门 ……61
4.2.2　与门 ……62
4.2.3　或门 ……62
4.2.4　异或门 ……63
4.2.5　与非门和或非门 ……63
4.2.6　门处理回顾 ……64
4.2.7　具有更多输入的门 ……64

4.3 门的构造……65
4.4 电路……66
4.4.1 组合电路……67
4.4.2 加法器……69
4.4.3 多路复用器……71
4.5 存储器电路……71
4.6 集成电路……72
4.7 CPU芯片……73
小结……73
道德问题：电子邮件隐私权……73
练习……74
思考题……76
第5章 计算部件……77
5.1 独立的计算机部件……77
5.2 存储程序的概念……79
5.2.1 冯·诺伊曼体系结构……80
5.2.2 读取–执行周期……84
5.2.3 RAM和ROM……85
5.2.4 二级存储设备……86
5.2.5 触摸屏……89
5.3 非冯·诺伊曼体系结构……90
小结……91
道德问题：生物信息学研究和deCODE Genetics公司的案例……92
练习……93
思考题……94

第四部分 程序设计层

第6章 问题求解和算法设计……95
6.1 问题求解……95
6.1.1 如何解决问题……96
6.1.2 应用Polya的问题求解策略……99
6.2 算法……99
6.2.1 计算机问题求解……99
6.2.2 执行算法……101
6.2.3 开发算法……102
6.3 伪代码……102
6.3.1 执行一个伪代码算法……103
6.3.2 伪代码的功能……104
6.3.3 伪代码示例……106
6.4 自顶向下设计方法……108
6.4.1 一个通用的实例……109
6.4.2 一个计算机实例……111
6.4.3 方法总结……113
6.4.4 测试算法……114
6.5 面向对象方法……114
6.5.1 面向对象……114
6.5.2 设计方法……115
6.5.3 一个通用的实例……117
6.5.4 一个计算机实例……118
6.6 几个重要思想……120
6.6.1 信息隐蔽……120
6.6.2 抽象……121
6.6.3 事物命名……122
6.6.4 程序设计语言……122
6.6.5 测试……123
小结……123
道德问题：计算机专业人员许可……124
练习……124
思考题……126
第7章 低级程序设计语言……127
7.1 计算机操作……127
7.2 抽象的分层……128
7.3 机器语言……128
7.4 一个程序实例……133
7.4.1 问题和算法……133
7.4.2 程序……134
7.5 汇编语言……137
7.5.1 Pep/7汇编语言……138
7.5.2 伪代码操作……138
7.5.3 “Hello”程序的汇编语言版本……139
7.5.4 一个新程序……140
7.5.5 具有分支的程序……142
7.5.6 具有循环的程序……144
7.6 其他重要思想……145
7.6.1 抽象……145
7.6.2 测试……146
7.6.3 测试计划实现……146

小结 ……147
道德问题：软件盗版和版权 ……148
练习 ……148
思考题 ……150
第8章 高级程序设计语言 ……151
8.1 翻译过程 ……151
8.1.1 编译器 ……151
8.1.2 解释器 ……152
8.2 程序设计语言的范型 ……154
8.3 命令式语言的功能性 ……155
8.3.1 布尔表达式 ……155
8.3.2 强类型化 ……156
8.3.3 输入/输出结构 ……159
8.3.4 控制结构 ……160
8.3.5 复合数据类型 ……172
8.4 面向对象语言的功能性 ……175
8.4.1 封装 ……175
8.4.2 继承 ……176
8.4.3 多态性 ……176
小结 ……177
道德问题：开源软件的发展 ……178
练习 ……179
思考题 ……180
第9章 抽象数据类型和算法 ……181
9.1 抽象数据类型 ……181
9.2 实现 ……182
9.2.1 基于数组的实现 ……182
9.2.2 链式实现 ……183
9.3 列表 ……185
9.3.1 列表的基本操作 ……185
9.3.2 其他列表操作 ……188
9.4 排序 ……188
9.4.1 选择排序 ……189
9.4.2 冒泡排序 ……190
9.4.3 快速排序 ……191
9.5 二分检索法 ……195
9.6 栈和队列 ……196
9.6.1 栈 ……197
9.6.2 队列 ……198
9.6.3 实现 ……198
9.7 树 ……199
9.7.1 二叉树 ……199
9.7.2 二叉检索树 ……200
9.7.3 其他操作 ……203
9.7.4 图 ……204
9.8 程序设计库 ……204
小结 ……205
道德问题：使用计算机的恶作剧和欺诈行为 ……205
练习 ……206
思考题 ……208

第五部分 操作系统层

第10章 操作系统 ……209
10.1 操作系统的角色 ……209
10.1.1 内存、进程和CPU管理 ……211
10.1.2 批处理 ……211
10.1.3 分时操作 ……212
10.1.4 其他OS要素 ……213
10.2 内存管理 ……213
10.2.1 单块内存管理 ……214
10.2.2 分区内存管理 ……215
10.2.3 页式内存管理 ……216
10.3 进程管理 ……218
10.3.1 进程状态 ……218
10.3.2 进程控制块 ……219
10.4 CPU调度 ……219
10.4.1 先到先服务 ……220
10.4.2 最短作业优先 ……220
10.4.3 循环调度法 ……221
小结 ……222
道德问题：数字版权管理和关于Sony公司的根目录案件的争论 ……223
练习 ……223
思考题 ……226
第11章 文件系统和目录 ……227
11.1 文件系统 ……227
11.1.1 文本文件和二进制文件 ……228

11.1.2 文件类型 ······228
11.1.3 文件操作 ······229
11.1.4 文件访问 ······230
11.1.5 文件保护 ······231
11.2 目录 ······232
11.2.1 目录树 ······232
11.2.2 路径名 ······234
11.3 磁盘调度 ······236
11.3.1 先到先服务磁盘调度法 ······237
11.3.2 最短寻道时间优先磁盘调度法 ······237
11.3.3 SCAN磁盘调度法 ······238
小结 ······238
道德问题：垃圾邮件 ······239
练习 ······240
思考题 ······241

第六部分 应用程序层

第12章 信息系统 ······243
12.1 信息管理 ······243
12.2 电子制表软件 ······244
12.2.1 电子数据表公式 ······246
12.2.2 循环引用 ······249
12.2.3 电子数据表分析 ······249
12.3 数据库管理系统 ······250
12.3.1 关系模型 ······251
12.3.2 关系 ······253
12.3.3 结构化查询语言 ······254
12.3.4 数据库设计 ······255
12.4 信息安全 ······256
12.4.1 机密性、完整性和可用性 ······256
12.4.2 密码学 ······257
小结 ······259
道德问题：加密 ······260
练习 ······261
思考题 ······262
第13章 人工智能 ······263
13.1 思维机 ······263
13.1.1 图灵测试 ······264
13.1.2 AI问题的各个方面 ······265
13.2 知识表示 ······265
13.2.1 语义网 ······266
13.2.2 检索树 ······268
13.3 专家系统 ······270
13.4 神经网络 ······272
13.4.1 生物神经网络 ······272
13.4.2 人工神经网络 ······273
13.5 自然语言处理 ······274
13.5.1 语音合成 ······275
13.5.2 语音识别 ······276
13.5.3 自然语言理解 ······276
13.6 机器人学 ······277
13.6.1 感知-规划-执行范型 ······277
13.6.2 包孕体系结构 ······280
13.6.3 物理部件 ······281
小结 ······281
道德问题：HIPAA（健康保险携带和责任法案） ······282
练习 ······283
思考题 ······284
第14章 模拟、图形学和其他应用程序 ······285
14.1 什么是模拟 ······285
14.1.1 复杂系统 ······286
14.1.2 模型 ······286
14.1.3 构造模型 ······286
14.1.4 排队系统 ······287
14.1.5 气象模型 ······290
14.1.6 其他模型 ······293
14.1.7 必要的计算能力 ······293
14.2 计算机图形学 ······294
14.2.1 光的工作原理 ······295
14.2.2 物体形状 ······296
14.2.3 光模拟 ······296
14.2.4 复杂对象的建模 ······297
14.2.5 让物体动起来 ······302
14.3 嵌入式系统 ······303
14.4 电子商务 ······303

14.5 计算机安全 ……304
14.5.1 恶意代码 ……305
14.5.2 安全攻击 ……305
小结 ……307
道德问题：入侵大学的计算机系统，查询录取程序中某人的录取状态 …307
练习 ……308
思考题 ……309

第七部分 通 信 层

第15章 网络 ……311
15.1 连网 ……311
15.1.1 网络的类型 ……312
15.1.2 Internet连接 ……314
15.1.3 包交换 ……316
15.2 开放式系统和协议 ……317
15.2.1 开放式系统 ……317
15.2.2 网络协议 ……318
15.2.3 TCP/IP ……318
15.2.4 高层协议 ……319
15.2.5 MIME类型 ……320
15.2.6 防火墙 ……320
15.3 网络地址 ……321
小结 ……323
道德问题：无所不在的计算 ……324
练习 ……325
思考题 ……326
第16章 万维网 ……327
16.1 Web简介 ……327
16.1.1 搜索引擎 ……329
16.1.2 即时消息 ……329
16.1.3 博客 ……329
16.1.4 cookie ……330
16.2 HTML ……330
16.2.1 基本的HTML格式 ……333
16.2.2 图像和链接 ……333
16.3 交互式Web页 ……335
16.3.1 Java小程序 ……335
16.3.2 Java服务器页 ……336
16.4 XML ……337
小结 ……339
道德问题：写博客 ……340
练习 ……341
思考题 ……342

第八部分 总 结

第17章 计算的限制 ……345
17.1 硬件 ……345
17.1.1 算术运算的限制 ……345
17.1.2 部件的限制 ……350
17.1.3 通信的限制 ……350
17.2 软件 ……351
17.2.1 软件的复杂度 ……352
17.2.2 当前提高软件质量的方法 ……352
17.2.3 臭名昭著的软件错误 ……355
17.3 问题 ……357
17.3.1 算法比较 ……357
17.3.2 图灵机 ……362
17.3.3 停机问题 ……364
17.3.4 算法分类 ……365
小结 ……367
道德问题：深度链接 ……367
练习 ……368
思考题 ……369
参考文献 ……370

第一部分 基 础 篇

第1章 全 景 图

这本书将带你游历计算世界，采用自底向上、由内到外的方式探讨计算机如何运作，它们可以做什么以及如何做。计算机系统就像一个交响乐团，把许多不同的元素组织在一起，构成了一个整体，但这个整体的功能却远远大于各个部件的功能总和。这一章综述了我们要在书中慢慢剖析的各个部件，从历史的角度来观察它们，提供了一个计算机系统的全景图。

硬件、软件、程序设计、网上冲浪和电子邮件这些术语都是你耳熟能详的。虽然有些人能够精确地定义这些与计算机相关的术语，但是其他人则对它们只有一个模糊的、直觉的概念。这一章则一视同仁，列出了通用的计算机术语，而且为我们深入探讨计算领域搭建了平台。

目标

学完本章之后，你应该能够：

- 描述计算机系统的分层。
- 描述抽象的概念以及它与计算的关系。
- 描述计算机硬件和软件的历史。
- 描述计算机用户转换的角色。
- 区分系统程序员和应用程序员。
- 区分计算工具和计算学科。

1.1 计算系统

在本书中，我们将探讨计算系统的方方面面。注意，我们使用的术语是计算系统，而不是计算机系统。计算机是一种设备，而**计算系统**则是一种动态实体，用于解决问题以及与它所处的环境进行交互。计算系统由硬件、软件和它们管理的数据构成。**计算机硬件**是构成机器及其附件（包括机箱、电路板、芯片、电线、硬盘驱动器、键盘、显示器、打印机，等等）的物理元件集合。**计算机软件**是提供计算机执行的指令的程序集合。计算机系统的核心是它管理的信息。如果没有数据，硬件和软件都毫无用处。

本书的基本目标有三个：

- 让你扎实、概括地理解计算系统是如何运作的。
- 让你理解与欣赏现代计算系统的进化。
- 给你足够的关于计算的信息，来决定是否深入探讨这个主题。

这一节剩余的部分解释了如何把计算机系统分成抽象层以及每一层扮演的角色。接下来

的一节把计算硬件和软件的开发置于历史背景中。本章的结尾讨论了计算工具和计算学科。

> **计算系统**（computing system）：通过交互解决问题的计算机硬件、软件和数据。
> **计算机硬件**（computer hardware）：计算系统的物理元件。
> **计算机软件**（computer software）：提供计算机执行的指令的程序。

1.1.1 计算系统的分层

计算系统就像一个洋葱，由许多层构成。每个分层在整个系统设计中都扮演一个特定的角色。计算系统的分层如图1-1所示，它们构成了本书的基本结构。在探讨计算系统的各个方面时，我们将不时地回顾这个全景图。



图1-1 计算系统的分层

你可能不会像咬苹果那样咬洋葱，但是可以把它分割成同心环。同样地，在这本书中，我们把计算分层逐个地从计算系统中剥离出来，每次只探讨一个分层。每个分层自身就不那么复杂了。事实上，我们指出了一台计算机真正所做的只是非常简单的任务，它盲目快速地执行这些任务，根本不知道可以把许多简单的任务组织成较大的复杂任务。当把各个计算机分层组织在一起，让它们各自扮演自己的角色，这种简单组合产生的结果却是惊人的。

让我们简单地讨论一下每个分层，并且说明在本书的什么地方会详细讨论它们。我们讨论的顺序是从内到外，也称为自底向上。

最内层的信息层反映了在计算机上表示信息的方式，它是一个纯概念层。计算机上的信息采用二进制数字1和0管理。所以，要理解计算机处理技术，首先必须理解二进制数制以及它与其他数制（如人们日常使用的十进制数制）的关系。然后介绍了如何获取多种类型（如数字、文本、图像、音频和视频）的信息，以及如何用二进制格式表示它们。第2章和第3章探讨了这些问题。

接下来的硬件层由计算机系统的物理硬件组成。计算机硬件包括的设备有门和电路，它们都按照基本原理控制电流。正是这些核心电路，使专用的元件（如计算机的中央处理器CPU和存储器）得以运转。第4章和第5章详细讨论了这些论题。

程序设计层负责处理软件、用于实现计算的指令以及管理数据。程序有多种形式，可以在许多层面上执行，由各种语言实现。尽管程序设计问题多种多样，但是它们的目的是相同

的，即解决问题。第6～9章探讨了许多与程序设计和数据管理相关的问题。

每台计算机都用操作系统（OS）管理计算机的资源。诸如Windows XP、Linux或Mac OS这样的操作系统可以使我们与计算机系统进行交互，管理硬件设备、程序和数据间的交互方式。了解操作系统为我们做了什么，通常是理解计算机的关键。第10章和第11章讨论了这些问题。

前面（内部）的分层重点在于使计算机系统运转，而应用层的重点则是用计算机解决真实世界的问题。我们通过运行应用程序在其他领域利用计算机的能力，例如设计一个建筑或打游戏。领域专用的计算机软件工具范围广大，涉及计算学的几个子学科，如信息系统、人工智能和仿真。第12章、13章和14章讨论了应用程序系统。

计算机不再只是某个人桌面上的孤立系统。我们使用计算机技术进行通信，通信层是计算系统操作的基础层。计算机被连接到网络上，以共享信息和资源。Internet逐渐演化成了全球性的网络，所以利用计算技术，可以与地球上的任何地方通信。World Wide Web使通信变得相对容易，它从根本上改变了计算机的使用价值，即使一般大众也能使用它。第15章和第16章讨论了这些有关计算通信的重要论题。

本书的大部分章节都是介绍计算机能够做什么以及如何做的。我们最终讨论了计算机不能做什么，或者至少不能做得很好。计算机在表示信息方面有固有的缺陷，程序设计只能尽可能地改善这一点。此外，还有一些问题是根本不能解决的。第17章分析了计算机的这些缺陷。

有时，我们很容易掌握细节，但却失去了全局观念。在阅读本书的过程中，请记住计算系统的全景图。每一章的首页都会提醒你处于计算系统的哪一个分层。所有的细节都只是为了给一个大整体贡献一个特定部分。每前进一步，你都会为它们如此精妙而吃惊不已。

1.1.2 抽象

我们刚才分析的计算系统的层次是抽象的一种例子。所谓**抽象**，是一种心理模型，是一种思考事情的方式，它删除或隐藏了复杂的细节。抽象只保留实现目标所必需的信息。当我们与计算机的一个分层打交道时，没有必要考虑其他分层。例如，在编写程序时，我们不必关心硬件是如何执行指令的。同样地，在运行程序时，我们也不必关心程序是如何编写的。

> **抽象**（abstraction）：删除了复杂细节的心理模型。

大量的实验表明，人在短期记忆中可以同时管理大约7条（根据个人情况，增加或减少2条）信息，这称为Miller定律，是Miller这位心理学家第一个研究的。[1]（注：此处的上角标数字表明有相应的参考文献，具体信息见书末的“参考文献”部分，全书同。）当我们需要其他信息时，可以得到它，但当我们集中于一条新信息时，其他的信息就会退回二级状态。

这个概念与变戏法的人能够同时在空中保持的球数是相似的。人的智力只能同时玩7个球，当我们拾起一个新球时，必须抛掉另一个球。虽然7看来是个小的数字，但关键在于每个球可以表示一种抽象，或者一大块信息。也就是说，我们抛的每个球都可以表示一个复杂的论题，只要我们将它看作一种想法即可。

我们的日常生活中充满了抽象。例如，要把一辆车开到商店去，我们不需要知道车是如何运转的。也就是说，我们根本不必详细地知道引擎是如何工作的。你只需要知道一些基础知识，即如何与车互动以及如何操作踏板、手柄和方向盘，甚至不必同时考虑这几个方面。请参阅图1-2。

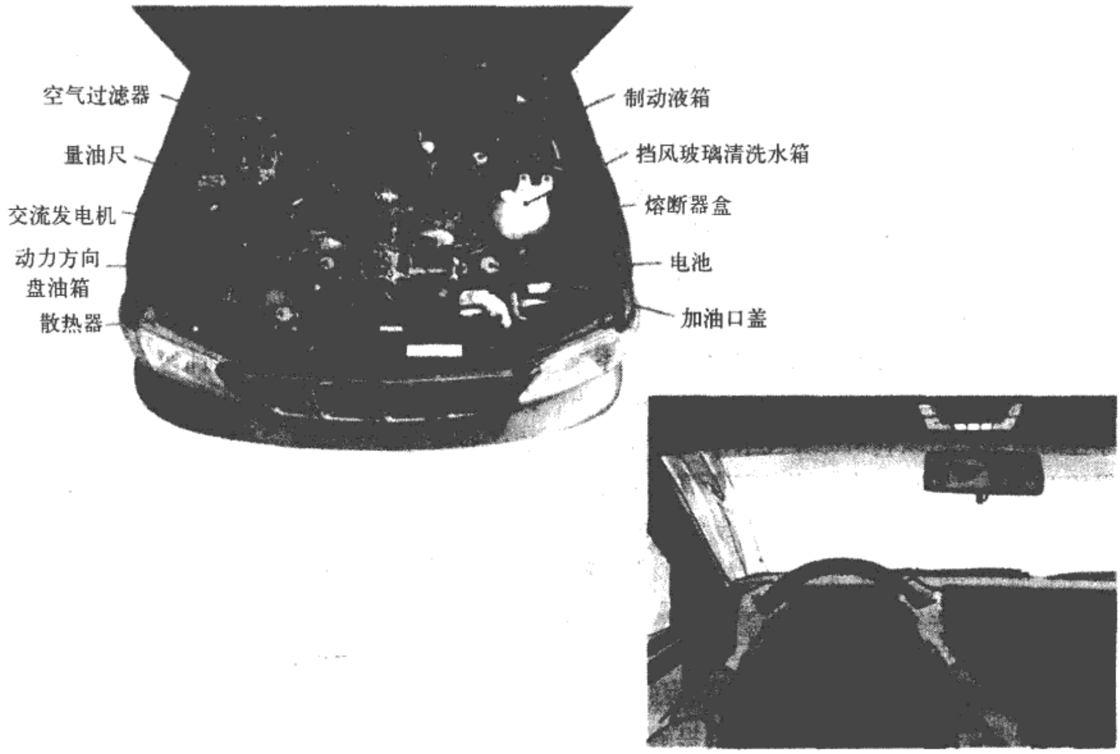


图1-2 汽车引擎和它的抽象

即使我们知道引擎是如何工作的，在开车时也不必考虑它。请想象一下，如果在开车时，我们必须不断地想着火花塞是如何点燃燃料从而驱动活塞推动曲柄轴的，那么哪儿也去不了。一部汽车太复杂，我们不能同时关注它的所有方面。这些技术细节就像变戏法时抛起的球，同时抛起所有技术细节就太多了。但是，如果能够把汽车抽象成较小的规模，使我们能与之交互，那么就可以将它作为一个实体处理。此时，无关的细节将被忽略。

顾名思义，抽象艺术是另一种抽象的例子。一幅抽象画确实表示某些东西，但绝不会陷于事实细节的泥淖。看看图1-3所示的抽象画，标题为Nude Descending a Staircase（下楼梯的裸女）。你只能看到一个女人或楼梯的迹象，因为画家对这个女人或这个楼梯的精确细节并不感兴趣。这些细节与画家的创作意图无关。事实上，现实的细节反而会妨碍那些画家认为重要的主题。

抽象是计算的关键。计算系统的分层表现了抽象的概念。此外，抽象还以各种形式出现在各个分层中。事实上，在我们接下来要探讨的计算系统的整个进化过程中，都有抽象的影子。



图1-3 抽象画Nude Descending a Staircase（下楼梯的裸女）

1.2 计算的历史

计算的历史十分悠久，可以解释为什么计算系统是今天这个样子。这一节讲述的故事中的人物和事件为我们开创了现在的天地，为开启激动人心的未来奠定了基础。我们分别描述了计算硬件和软件的历史，因为它们对计算系统进化为我们所用的层次模型有着不同的影响。

我们介绍历史时采用的是叙述性方式，没有正式地定义任何概念。在接下来的几章中，我们将定义这些概念，并且详细地研究它们。

1.2.1 计算硬件的简史

辅助人们进行各种计算的设备自古就有，迄今为止，它们还在不断进化中。让我们来简单浏览一下计算硬件的历史。

早期历史

许多人都相信位于英国的Stonehenge石群是早期的日历或星象观测台。出现在公元前16世纪的算盘是一种记录数值的工具，人们可以用它进行基本的数学运算。

Stonehenge石群仍然是一个谜



英国Salisbury平原附近庄严耸立的石群Stonehenge数个世纪以来一直吸引着人们的注意。很多人相信Stonehenge是从公元前2180年开始竖立的，历时几个世纪，才形成现在的规模。尽管存在大量的理论基础，但是它的目的仍然是个谜。在夏至这天，朝阳将出现在一块主要巨石之后，这使人们猜想阳光在这块石头上被抵消了，导致了关于Stonehenge的早期理论，即它是一个神庙。另一种理论出现在20世纪中期，认为Stonehenge可能是一种天文历法，使月亮和太阳排成了一条直线。第三种理论则认为Stonehenge用于预言日食。无论为什么建造了Stonehenge，它所处位置的神秘特质足以挑战任何解释。

文字之前的记数法

人类花了4000年时间才把三维代币完全简化为书写符号。这个过程可以追溯至公元前7500年，当时农夫们利用很多块形状各异的土块作为计数器，帮助自己计算资产。例如，圆锥形的土块表示少量的谷物，球形的土块表示大量的谷物，圆柱形的土块表示一只动物。四块圆锥形的土块表示四份少量的谷物。从Palestine、Anatolia、Syria、Mesopotamia和Iran发现了大约8000块这样的代币。

约公元前3500年，古希腊城邦兴起后，执政者开始使用土球作为封套盛放代币，有些土球被盖上了印章，标识了它们盛放的代币。下一次变化发生在公元前3300～3200年间，记录员开始只用土球上的印章进行计量，而忽略了代币本身。因此，把三维的代币简化成书写符号用了将近4000年的时间。

大约公元前3100年，开始使用铁笔绘制代币，而不再把代币符号盖在桌子上。这一变化打破了符号与对象之间的一一对应关系。十坛油用一坛油和一个代表10的符号表示。虽然没有创建新的符号表示抽象数字，但旧的符号有了新的含义。例如，圆锥形符号最初表示的是少量的谷物，当时变成了代表“1”的符号；球形（大量谷物）变成了“10”

的代表。这样，33坛油就可以用10＋10＋10＋1＋1＋1和油的代表符号来表示。

一旦创建了抽象数字，资产的符号和数字的符号就会发展成另一种形式。从而，文字是从记数法演化来的。

——摘自Denise Schmandt-Berrerat的《Signs of Life》，Odyssey，2002年1-2月，第6、7和63页

17世纪中叶，法国数学家Blaise Pascal建造并出售了一种齿轮驱动的机械机器，它可以执行整数的加法和减法运算。17世纪末，德国数学家Gottfried Wilhelm von Leibniz建造了第一台能够进行四种整数运算（加法、减法、乘法和除法）的机械设备。遗憾的是，当时的机械齿轮和操作杆的水平有限，使Leibniz机的结果不那么可信。

18世纪晚期，Joseph Jacquard发明了Jacquard织布机。这种织布机利用一套穿孔卡片来说明需要什么颜色的线，从而控制了纺织图案。尽管Jacquard织布机不是一种计算设备，但是它第一次使用了穿孔卡片这种输入形式。

未知的梦想

"谁能预见这种发明的重要性？分析机编织的代数模式就像Jacquard织布机织出的花朵与树叶一样。这种机器也许还可以编写复杂精美的音乐片段，音乐的复杂度和音域都毫无限制呢。"

Ada，Lovelace伯爵夫人，1843年[2]

计算硬件的下一步重大进展直到19世纪才出现，这一次是由英国数学家Charles Babbage发明的，他称之为分析机。他的设计太过复杂，以至于当时的技术水平不能建造那样的机器，所以他的发明根本就没有实现。但是，在他构想的设计中，却包括许多现代计算机的重要部件。他的设计中第一次出现了内存，这使中间值不必再重新输入。此外，他的设计还包括数字输入和机械输入法，采用了与Jacquard织布机使用的穿孔卡片相同的方式。

Lovelace伯爵夫人Ada Augusta是计算历史上的传奇人物。Ada是英国诗人Lord Byron的女儿，是一位杰出的数学家。她对Babbage的分析机非常感兴趣，扩充了他的想法（同时修改了他的一些错误）。Ada以第一位程序设计员著称。循环的概念（即一系列重复执行的指令）也归功于她。美国国防部广泛使用的Ada程序设计语言即是以她的名字命名的。

19世纪晚期和20世纪初，计算系统迅速发展起来。William Burroughs制造并销售了一台机械加法机。Herman Hollerith博士发明了第一台机电式制表机，从穿孔卡片读取信息。他的设备从根本上改变了美国每十年举行一次的人口普查。后来，Hollerith博士创建了当今著名的IBM公司。

第一位程序设计员Ada Lovelace[3]



1815年12月10日（George Boole也出生在这一年）Anna Isabella（Annabella） Byron和George Gordon，Lord Byron的女儿Augusta Ada Byron出世了。当时，Byron以他的诗歌和疯狂的、离经叛道的行为闻名于英国。Byron夫妇的婚姻关系从开始就很紧张，Ada出生后不久，Annabella就离开了Byron。两人于1816年4月签署了离婚协议。Byron离开了英国，从此再没有回来。因为不能见到自己的女儿，

他的余生充满了深深的遗憾。他在给Ada的一封信中写道：

我看不到你，我听不到你。

但是没有人像我这样关注着你。

他36岁时死于希腊，在去世之前，他大喊：

噢，我可怜可亲的孩子！我亲爱的Ada!

上帝啊，我能见到她吗？！

与此同时，Annabella最终成为了一位女伯爵，身兼数学家和诗人的称号，她负责对Ada的抚养和教育。最初，Annabella自己指导Ada学习数学，但是，Ada的数学天赋很快显露出来，她需要更广博的辅导。Augustus DeMorgan，这位布尔算术基本定理的发现者之一对Ada进行了进一步训练。8岁时，Ada表现出对机械设备的浓厚兴趣，建造了复杂的模型船。

当Ada 18岁时，她拜访了Mechanics Institute，聆听Dionysius Lardner博士关于差分机的讲座，差分机是Charles Babbage建造的一种机械计算机。据说，在见到Babbage的机器之前，Ada是教室中唯一一个立刻理解了这种机器的工作原理并且认识到它的价值的人。Ada和Charles Babbage从此成了终生的朋友。她与Babbage一起工作，帮助他记录设计方案，翻译他的工作记录，并且为他的机器开发程序。事实上，现在普遍认为Ada是历史上第一位计算机程序设计员。

当Babbage设计他的分析机时，Ada预见到这种机器的能力将远远超过算术计算，它将成为一种通用的符号操作装置。她甚至提出这种设备最终可以把和声与乐曲编写成程序，从而制作出"科学"之声。实际上，Ada是在150年前预见了人工智能这一领域。

1842年，Babbage在意大利的都灵举办了一系列有关分析机的讲座。一位参与者Luigi Menabrea被Babbage的讲座深深地打动了，于是编写了一份讲座记录。Ada于27岁时决定把这份记录翻译成英文，并且加入几点自己对这部机器的注释。最后，她的注释相当于原材料的两倍长，这份文献"分析机概要"成为了这一领域的权威著作。

从Ada的信件中可以看出，这些注释完全出自Ada之手，Babbage有时志愿当她的编辑，但Ada并不领情。在Ada给Babbage的一封信中写道：

你更改了我的注释，这让我非常生气。我一直愿意自己对它进行必要的修改，但是我不能忍受其他人乱动我的文句。

当Ada嫁给了Lord William Lovelace后，她得到了Lovelace伯爵夫人的头衔。这对夫妇有三个孩子，都由Ada的母亲抚养，Ada则继续她的数学工作。尽管她的丈夫支持她的事业，但在那个年代，从事这种工作的妇女被看作是可耻的，就像她的父亲那样惹人厌烦。

Ada死于1852年，仅比按照Babbage的设计在瑞典建造出第一台可以运行的差分机早一年。和她的父亲一样，Ada只活了36年，但即使如此，他们的生活是完全不同的，毫无疑问，Ada钦佩她的父亲，从他那些惊世骇俗和桀骜不逊的天性中汲取了灵感。最后，Ada要求把自己安葬在家族庄园中父亲的坟墓旁。

1936年，一种理论得以发展，本质上它与硬件毫无关系，但它对计算机科学产生了深远的影响。英国数学家Alan M. Turing发明了一种抽象数学模型图灵机，为计算理论的主要领域奠定了基础。计算机科学这一领域荣誉最高的奖就是图灵奖（相当于数学领域的菲尔丁奖章

或其他科学领域的诺贝尔奖），以Alan Turing的名字命名。最近有一部百老汇音乐剧演绎了他的一生。分析图灵机的功能是所有学习计算机科学的学生的理论学习的一部分。

到第二次世界大战爆发时，已经有几台计算机处于设计和建造中。Harvard Mark I和ENIAC是当时最著名的两台机器。图1-4所示的是ENIAC。John von Neumann是ENIAC这个项目的顾问，之后他开始致力于另一台著名机器EDVAC的建造，这台机器完成于1950年。1951年，美国人口普查局收到了第一台商业计算机UNIVAC I。UNIVAC I是第一台用于统计美国总统大选结果的计算机。

UNIVAC I的出现结束了以算盘为开端的计算早期历史。UNIVAC I实现了建造能快速操作数字的设备的梦想。探索结束了吗？在那个年代许多专家预言只有少数计算机能够满足人类的计算需求。但他们没有意识到，快速运算大量数据的能力可以从根本上改变数学、物理、工程学和经济这些领域的本性。也就是说，计算机使这些专家对“需要计算什么”的看法变成了无稽之谈。[4]

1951年后，计算机被越来越广泛地用来解决各个领域中的问题。从那时起，探索的重点不仅在于建造更快、更大的计算设备，而且在于开发能让我们更有效地使用这些设备的工具。从这时开始，计算硬件的历史基于它们所采用的技术被划分为几个时代。

第一代（1951～1959）

第一代（大约从1951年到1959年）商用计算机使用*真空管*存储信息。图1-5展示了一个真空管，它会大量生热，不是非常可靠。使用真空管的机器需要重型空气调节装置以及不断的维修。此外，它们还需要巨大的专用房间。



图1-4 第二次世界大战时期的计算机ENIAC

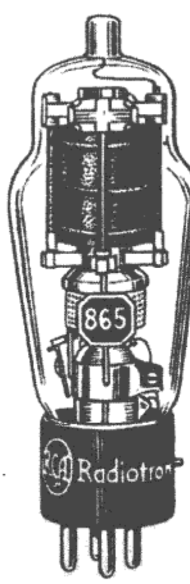


图1-5 真空管

第一代计算机的主存储器是在读/写臂下旋转的*磁鼓*。当被访问的存储器单元旋转到读/写臂之下时，数据将被写入这个单元或从这个单元中读出。

输入设备是一台读卡机，可以阅读IBM卡（由Hollerith卡演化而来）上的孔。输出设备是穿孔卡片或行式打印机。在这一代将要结束时，出现了*磁带驱动器*，它比读卡机快得多。磁

带是顺序存储设备，也就是说，必须按照线性顺序访问磁带上的数据。

计算机存储器外部的存储设备叫做辅助存储设备。磁带是一种辅助存储设备。输入设备、输出设备和辅助存储设备一起构成了外围设备。

第二代（1959～1965）

晶体管（John Bardeen、Walter H. Brattain和William B. Shockley为此获得了诺贝尔奖）的出现标志着第二代商用计算机的诞生。晶体管代替真空管成为了计算机硬件的主要部件。图1-6展示了一个晶体管，它比真空管更小、更可靠、更快、寿命更长也更便宜。

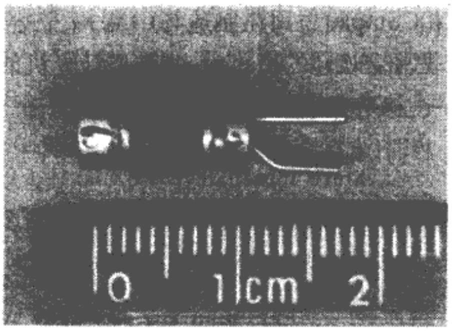


图1-6 晶体管，它替代了真空管

第二代计算机中还出现了即时存取存储器。访问磁鼓上的信息时，CPU必须等待读/写臂旋转到正确的位置。第二代计算机使用磁芯作为存储器，这是一种微小的环形设备，每个磁芯可以存储一位信息。这些磁芯由电线排成一列，构成存储单元，存储单元组合在一起构成了存储单位。由于设备是静止不动的，而且是用电力访问的，所以能够即时访问信息。

磁盘是一种新的辅助存储设备，也出现在第二代计算机中。磁盘比磁带快，因为使用数据项在磁盘上的位置就可以直接访问它。访问磁带上的一个数据项时，必须先访问这个数据项之前的所有数据，而磁盘上的数据都有位置标识符，我们称之为地址。磁盘的读/写头可以被直接送到磁盘上存储所需的信息的特定位置。

第三代（1965～1971）

在第二代计算机中，晶体管和其他计算机元件都被手工集成在印刷电路板上。第三代计算机的特征是集成电路（IC），一种具有晶体管和其他元件以及它们的连线的硅片。集成电路比印刷电路小，它更便宜、更快并且更可靠。Intel公司的奠基人之一Gordon Moore注意到从发明IC起，一个集成电路板上能够容纳的电路的数量每年增长一倍，这就是著名的摩尔定律。[5]

晶体管也被应用在存储器构造中。每个晶体管表示一位信息。集成电路技术允许用晶体管建造存储板。辅助存储设备仍然是必需的，因为晶体管存储器不稳定，也就是说，断电之后，所有的信息都将消失。

终端（带有键盘和屏幕的输入/输出设备）便是在这一代计算机中出现的。键盘使用户可以直接访问计算机，屏幕则可以提供立即响应。

第四代（1971～?）

大规模集成化是第四代计算机的特征。20世纪70年代早期，一个硅片上可以集成几千个晶体管，而80年代中期，一个硅片则可以容纳整个微型计算机。主存储设备仍然依赖芯片技术。在过去的40年中，每一代计算机硬件的功能都变得越来越强大，体积越来越小，花费也越来越少。Moore定律被改为芯片的集成度每18个月增长一倍。

20世纪70年代末，词汇表中出现了个人计算机（PC）这个词。Apple、Tandy/Radio Shack、Atari、Commodore和Sun公司加入了早期计算机公司的行列，这些公司包括IBM、Remington Rand、NCR、DEC（Digital Equipment Corporation）、Hewlett-Packard、Control Data和Burroughs。在个人计算机革命的浪潮中，最为人称道的成功故事是关于Apple公司的。

工程师Steve Wozniak和中学生Steve Jobs共同创建了一个计算机工具包，而且把它从车库推向了市场，这就是Apple Computer这个拥有数十亿资产的公司的起源。

IBM PC是1981年面世的，之后，其他公司迅速制造了许多与之兼容的机器。例如，Dell和Compaq公司在制造与IBM PC兼容的PC方面取得了巨大的成功。Apple公司在1984年创建了非常受欢迎的Macintosh微型计算机的生产线。

从车库到财富500强

Steve Jobs和Steve Wozniak是少年时代的朋友，他们分别卖掉了自己的大众汽车和可编程的计算器，为他们的新计算机公司筹集资金。他们的第一单生意是销售50 Apple ls，即他们在车库中设计和建造的计算机。在短短的6年时间中，Apple公司跻身于世界财富500强，是这个声誉卓著的列表上最年轻的公司。

20世纪80年代中期，出现了更大型、功能更强大的机器*工作站*，它们通常用于商业用途，而不适用于个人。创造工作站的理念是为了把雇主自己的工作站放在一个桌面上。这些工作站由线缆连接在一起，或者说被*连网*了，以便它们彼此能够交互。引入了RISC（精简指令集计算机，reduced-instruction-set computer）体系结构后，工作站变得更加强大了。每台计算机都能理解一套指令，我们称这套指令为*机器语言*。传统机器（如IBM 370/168）的指令集有200多条指令。指令执行得非常快，但访问内存的速度却很慢，因此，特殊的指令更加有用。随着内存访问的速度越来越快，使用精简的指令集变得越来越诱人。Sun微系统公司于1987年制造出了采用RISC芯片的工作站。这种工作站的受欢迎程度说明了RISC芯片的可行性。我们通常称这些工作站为UNIX工作站，因为它们使用的操作系统是UNIX。

由于计算机仍在使用电路板，所以我们不能为这一代计算机画上休止符。但是，一些新事物已经出现了，对如何使用计算机造成了强烈的影响，它们必将开创一个新时代。摩尔定律被再次改写为下列说法："每18个月，计算机的功率会在同样的价格水平下增长一倍，或者以一半的价格可以购买同样的计算机功率"。[6]

并行计算

20世纪80年代末，尽管使用单处理器的计算机仍然盛行，但是新的机器体系结构出现了。使用*并行体系结构*的计算机依靠的是一套互相连接的中央处理器。

一种并行机器的组织结构是所有处理器共享同一个存储部件。另一种组织结构是每个中央处理器具有自己的本地内存，与其他处理器通过高速内部网进行通信。

*并行体系结构*提供了几种加快执行速度的方法。例如，把程序中的一步操作分成多个片段，在几个独立的处理器上同时执行这些程序片段。这种机器被称为SIMD（单指令多数据流，single-instruction, multiple-data-stream）计算机。第二种机器可以同时运行程序的不同部分。这种机器被称为MIMD（多指令多数据流，multiple-instruction, multiple-data-stream）计算机。

虽然把上百甚至上千个处理器组织在一台机器中有巨大的潜能，但是为这种机器进行程序设计的难度也很高。并行计算机的软件设计不同于一个计算机序列的软件设计。程序设计员必须重新思考解决问题的方法，利用并行性进行程序设计。

连网

20世纪80年代，多用户大型机的概念被小型机器连接成的网络代替，这些小型机器通过

连网共享打印机、软件和数据这些资源。1973年由Robert Metcalfe和David Boggs发明的以太网是一种廉价的同轴电缆和一套能够让机器互相通信的协议。1979年，DEC、Intel和Xerox公司都参与到以太网标准的制定中。

工作站的设计是为了连网，但是，直到1985年生产出了更高级的Intel芯片，才能够对个人计算机进行连网。1989年，Novell Netware用文件服务器把PC连接在一起。**文件服务器**是一台具有大容量的存储器以及强劲的输入/输出能力的PC。把数据和办公自动化软件放在服务器上，而不是在每个PC上放置一个副本，这样既达到了集中控制的目的，又给予了每台PC自主权。把工作站或PC连接成网络，就形成了LAN（局域网，local area network）。

我们知道，Internet是从ARPANET演化来的，ARPANET是美国政府从20世纪60年代开始资助的网络，由11个节点构成，集中分布在Los Angeles和Boston地区。与ARPANET和LAN一样，Internet使用**包交换**的方法共享信息。但是，Internet由分布在世界各地的不同网络组成，这些网络之间采用通用的TCP/IP（传输控制协议/网际协议，transmission-control protocol/internet protocol）协议通信。

> **什么是协议？**
>
> 协议是一套规定必须严格遵守的规则和（交互信息的）过程的代码。计算术语学借用这个词来表示计算机交互时使用的正确规定。

在《现代计算史》（A History of Modern Computing）一书中，Paul E. Ceruzzi对以太网和Internet的关系做了下列注解：

“如果20世纪90年代的Internet是信息高速公路，那么以太网就是支持它的慢车道，两者同等重要。Internet是由ARPA研究演化来的全球网络，在Xerox公司发明本地以太网前，Internet已经存在了。但是，在Internet盛行之前，以太网改变了办公室计算和个人计算的本性”。[7]

1.2.2 计算软件的简史

虽然计算机硬件可以启动，但是如果没有构成计算机软件的程序的指引，它们什么也做不了。了解软件进化的方式，对理解软件在现代计算系统中是如何运行的至关重要。

第一代软件（1951～1959）

第一代程序是用机器语言编写的。所谓机器语言，即内置在计算机电路中的指令。即使是对两个数字求和这样的小任务也要动用3条二进制指令，程序设计员必须记住每种二进制数字的组合表示什么。使用机器语言的程序设计员一定要对数字非常敏感，而且要非常细心。第一代程序员是数学家和工程师就毫不令人感到惊奇了。然而，用机器语言进行程序设计不仅耗时，而且容易出错。

由于编写机器代码非常乏味，有些程序设计员就开发了一些工具辅助程序设计。因此，第一代人工程序设计语言出现了。这些语言被称为**汇编语言**，它们使用助记忆码表示每条机器语言指令。

由于每个程序在计算机上执行时采用的最终形式都是机器语言，所以汇编语言的开发者还创建了一种**翻译程序**，把用汇编语言编写的程序翻译成用机器语言编写的。一种称为**汇编器**的程序将读取每条用助记忆码编写的程序指令，把它翻译成等价的机器语言。这些助记忆

码都是缩写码，有时难以理解，但它们比二进制数字串容易用得多。

那些编写辅助工具的程序设计员，简化了他人的程序设计，是最初的*系统程序员*。因此，即使在第一代计算机软件中，也存在编写工具的程序设计员和使用工具的程序员这样的分类。汇编语言是程序设计员和机器硬件之间的缓冲器。请参阅图1-7。即使是现在，如果高效代码是必需的，那么还是会用汇编语言编写程序。第7章详细探讨了一个机器代码和它对应的汇编语言的例子。

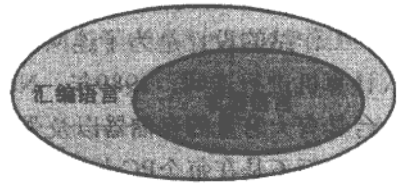


图1-7 第一代末期计算机语言的分层

第二代软件（1959～1965）

当硬件变得更强大时，就需要更强大的工具能有效地使用它们。当然，汇编语言是向正确的方向前进了一步，但是程序设计员还是必须记住单独的机器指令。第二代软件一定要由更强大的语言开发。使用*高级语言*，程序设计员就能够用类似于英语的语句编写指令。

第二代软件时期开发的两种语言，目前仍然在使用，它们是FORTRAN（为数字应用程序设计的语言）和COBOL（为商业应用程序设计的语言）。FORTRAN和COBOL的开发过程完全不同。FORTRAN最初是一种简单语言，经过几年附加特性后才形成一种高级语言。而COBOL则是先设计好，然后再开发的，形成之后就很少改动。

这一时期设计的另一种仍然在用的语言是Lisp。Lisp与FORTRAN和COBOL有极大的不同，而且没有被广泛接受，主要用于人工智能的应用程序和研究。Lisp的专用语是当今人工智能可用的语言之一，Scheme就是一种Lisp专用语，有些学校用它作为启蒙性的程序设计语言。

高级语言的出现加速了在多台计算机上运行同一个程序。每种高级语言都有配套的翻译程序，这种程序可以把高级语言编写的语句翻译成等价的机器码指令。最早时，高级语言的语句通常被翻译成汇编语言，然后这些汇编语句再被翻译成机器码。只要一台机器具有*编译器*这种翻译程序，就能够运行用FORTRAN或COBOL编写的程序。

在第二代软件末期，系统程序员的角色变得更加明显。系统程序员编写诸如汇编器和编译器这样的工具。使用这些工具编写程序的人，被称为*应用程序设计员*。随着包围硬件的软件变得越来越复杂，应用程序设计员离计算机硬件越来越远了。请参阅图1-8。

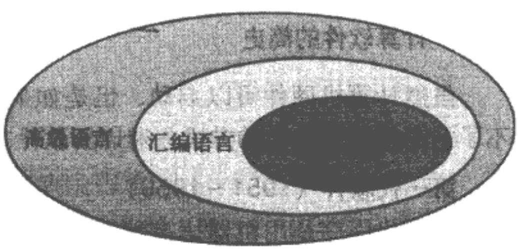


图1-8 第二代末期计算机语言的分层

第三代软件（1965～1971）

在第三代商用计算机时期，很显然，人们使计算机的处理速度放慢了。计算机在等待运算器准备下一个作业时，无所事事。解决方法是使所有计算机资源处于计算机的控制中，也就是说，要编写一种程序，决定何时运行什么程序。这种程序被称为*操作系统*。

在前两代软件时期，实用程序用于处理频繁执行的任务。*装入器*把程序载入内存，*连接器*则把大型程序连接在一起。第三代软件改进了这些实用程序，使它们处于操作系统的引导下。实用程序、操作系统和语言翻译程序（汇编器和编译器）构成了*系统软件*。

用作输入/输出设备的计算机终端的出现，使用户能够访问计算机，而高级的系统软件则使机器运转得更快。但是，从键盘和屏幕输入输出数据是个很慢的过程，比在内存中执行指

令慢得多。这就导致了如何利用机器越来越强大的能力和速度的问题。解决方法就是分时，即许多用户用各自的终端同时与一台计算机进行通信（输入和输出）。控制这一进程的是操作系统，它负责组织和安排各个作业。

对于用户来说，分时好像使他们有了自己的机器。每个用户都会被分配到一小段中央处理时间，在中央处理器服务于一个用户时，其他用户将处于等待状态。用户通常不会察觉还有其他用户。但是，如果同时使用系统的用户太多，那么等待一个作业完成的时间就会变得很明显。

在第三代软件中，出现了多用途的应用程序，用FORTRAN语言编写的社会科学统计程序包（Statistical Package for the Social Science, SPSS）就是这样的程序。SPSS具有一种专用的语言，用户使用这种语言编写指令，作为程序的输入。使用这种专用语言，即使不是经验丰富的程序设计员也可以描述数据，并且对这些数据进行统计计算。

起初，计算机用户和程序设计员是一体的。在第一代软件末期，为其他程序设计员编写工具的程序设计员的出现带来了系统程序设计员和应用程序设计员的区分。但是，程序设计员仍然是用户。在第三代软件中，系统程序员为其他人编写软件工具。计算机用户的概念骤然出现了，他们不再是传统意义上的程序员。

计算机控制的施肥器

施放适量氮肥以最大化玉米产量是经济学和环境学的一大挑战。氮肥太少，会导致产量低；氮肥太多则成本太高，而且会危及水的质量。科学家开发了一套系统，通过测量玉米幼苗的颜色，使用连接到计算机控制的氮肥施放器上的发光二极管（LED）传感器，来施放适量的氮肥。科学家希望这种方法既可以提高产量，又可以创造一个更健康的环境。

用户与硬件的距离逐渐加大。硬件演化成整个系统的一小部分。由硬件、软件和它们管理的数据构成的计算机系统出现了，如图1-9所示。虽然语言层还在加深，但是程序设计员们仍然在使用一些最内层的语言。如果要求一小段代码运行得尽可能快，占用的内存尽可能少，那么还是需要用汇编语言或机器语言编写这段代码。

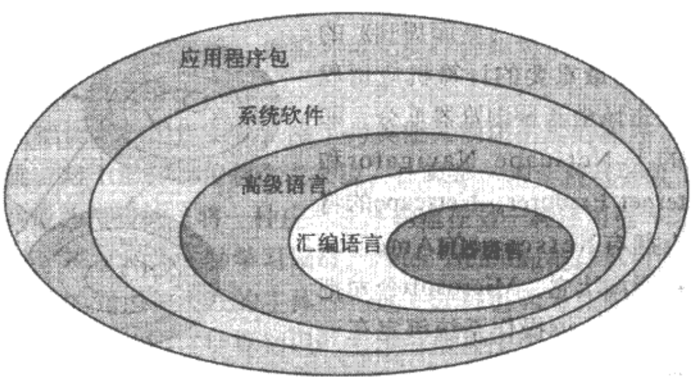


图1-9 包围硬件的软件分层仍然在增长

第四代软件（1971～1989）

20世纪70年代出现了更好的程序设计技术——*结构化程序设计方法*，一种有逻辑、有规则的程序设计方法。Pascal语言和Modula-2都是采用结构化程序设计的规则制定的。BASIC这

种为第三代机器设计的语言也被升级成了更具有结构性的版本。此外，还出现了C语言，使用这种语言，用户可以在高级程序中使用一些汇编语句。C++也是一种允许用户使用低级语句的结构化语言，它成为了业界的选择。

更好、更强大的操作系统也被开发出来了。AT&T公司作为研究工具开发的UNIX系统成了许多大学的标准设置。为IBM PC开发的PC-DOS系统和为了兼容开发的MS-DOS系统都成了个人计算机的标准系统。Macintosh机的操作系统引入了鼠标的概念和点击式的图形界面，因此彻底改变了人机交互的方式。

即使在附近的小店，都可以买到高品质的、价格合理的应用程序软件包。这些程序可以让一个没有计算机经验的用户实现一项特定的任务。三种典型的应用程序包是*电子制表软件*、*文字处理软件*和*数据库管理系统*。Lotus1-2-3是第一个商用电子制表软件，即使一个新手，也可以用它输入数据，对数据进行各种分析。WordPerfect是第一个文字处理软件，dBase IV是让用户存储、组织和提取数据的系统。

第五代软件（1990～今天）

第五代中有三个著名事件，即在计算机软件业具有主导地位的Microsoft公司的崛起、面向对象的设计和编程方法的出现以及万维网（World Wide Web）的普及。

在这一时期，Microsoft公司的Windows操作系统在PC市场占有显著优势。尽管WordPerfect仍在继续改进，但是Microsoft公司的Word成了最常用的文字处理软件。20世纪90年代中期，文字处理软件、电子制表软件、数据库程序和其他应用程序都被绑定在一个超集程序包中，这个程序包称为*办公套件*。

*面向对象的程序设计方法*成为大型程序设计项目的首选。结构化设计基于任务的层次划分，而面向对象的设计则基于数据对象的层次划分。Sun Microsystems公司为面向对象的编程方法设计的Java语言成为了C++语言的竞争对手。

1990年，日内瓦的CERN物理实验室的英国研究员Tim Berners-Lee希望创建一个全球Internet文档中心——*万维网*（World Wide Web），他为之创建了一套技术规则。除了这套规则，他还创建了格式化文档的HTML语言和让用户访问全世界站点上的信息的程序——浏览器，此时的浏览器还不成熟，只能显示文本。1993年，Marc Andreesen和Eric Bina发布了第一个能显示图形的浏览器Mosaic。美国《新闻周刊》的报道称“Mosaic将成为最重要的计算机应用程序”。[8] 目前的浏览器市场由两种浏览器瓜分，即（由Mosaic衍生的）Netscape Navigator和Microsoft公司的Internet Explorer。Netscape的将来很难预测，因为拥有Netscape的America Online公司可能会逐渐淘汰它。Microsoft公司把Internet Explorer与Windows操作系统绑定在一起，遭到了反托拉斯组织的强烈反对。

虽然Internet已经存在几十年了，但是万维网的出现，让使用Internet在世界范围内共享信息变得容易了（参阅图1-10）。

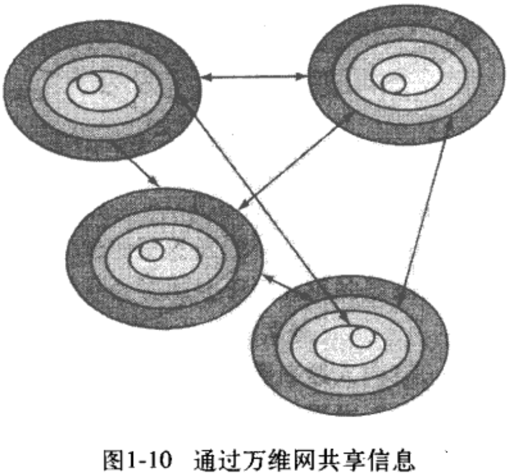


图1-10 通过万维网共享信息

20世纪80年代和90年代最重要的特征是用户概念的改变。首先出现的用户是程序设计员，

他们编写程序来解决自己或他人的问题。接下来出现的用户是系统程序员，他们为其他程序员编写越来越复杂的工具。20世纪70年代早期，应用程序员使用这些复杂的工具为非程序员编写应用程序。随着个人计算机、计算机游戏、教育程序和用户友好的软件包的出现，许多人成为了计算机用户。万维网的出现，使网上冲浪成了娱乐方式的一种选择，所以更多的人成了计算机用户。计算机用户可以是在学习阅读的婴儿，可以是在下载音乐的青少年，可以是在写论文的大学生，可以是在制定预算的家庭主妇，可以是在查找客户信贷记录的银行职员。我们所有的人都是计算机的用户。

在硬件和软件的历史简介中，我们把重点放在传统的计算机和计算系统上。与这些历史并驾齐驱的，是使用集成电路（或芯片）来运行或控制烤面包机、汽车、重病特别护理监控器和卫星的历史。这种计算技术称为*嵌入式系统*。虽然芯片不是我们要在本书中研究的真正意义上的计算机，但它们确实是过去55年中的技术革命的产物。

1.2.3 预言

我们用几个没有实现的关于计算机的预言来结束计算历史的简介：[9]

“*我认为存在大概5台计算机的世界市场。*”——Thomas Watson，*IBM公司主席*，1943

“*ENIAC有18 000个真空管，重达30吨。未来的计算机将只有1000个真空管，重量只有1.5吨。*”——*Popular Mechanics*，1949

“*朋友们，Mac平台将完全占领市场。*”——John C.Dvorak，*PC Magazine*，1998

“*没理由人人都想在家摆一台计算机。*”——Ken Olsen，*DEC公司总裁、主席及创始人*，1977

“*我预测Internet将成为一颗惊人的超新星，1996年则会彻底失败。*”——Bob Metcalfe，*3COM公司的创始人和发明家*，1995

1.3 计算工具和计算学科

在计算机软件历史简介一节中，我们指出了用户角色的不断改变。在第一代软件末期，用户被划分为两组，即开发使程序设计更简单的工具的系统程序员和使用这些工具的应用程序员。此后，应用程序员利用传统的软件工具开发大量专用的应用程序，如统计包、文字处理程序、电子制表软件、智能浏览器、虚拟环境和医疗诊断应用程序。而这些应用程序又由没有计算机背景的从业人员使用。

因此，到底谁在把计算机用作工具？除了为其他人创建工具的程序员之外，所有人都在使用计算机这个工具。对于那些工具制作者来说，计算是一种学科（低级工具），或者计算这种学科使他们的工具成为可行的（将一种应用程序构建在另一种应用程序之上）。

学科（discipline）被定义为一种学习领域。Peter Denning把计算机科学学科定义为“计算机专家在工作中使用的知识和实践的主体……这一学科也称为计算机科学和工程学、计算学或信息学”。[10] 他继续说明道，“计算知识的主体经常被描述为对算法过程的系统研究，包括算法的理论、分析、设计、有效性、实现和应用。隐藏在所有计算问题之下的基本问题是‘*什么可以被有效地自动操作？*’”

Denning认为每个从业人员需要四个领域的技巧。

- *算法思想*，即能够用按部就班的过程表示问题，从而解决它们。
- *表示法*，即用能被有效处理的方式存储数据。

- 程序设计，即把算法思想和表示法组织在计算机软件中。
- 设计，使软件满足一种用途。

关于计算学是一种数学学科，还是一种科学学科或工程学科，存在着长期的争论。计算学当然来源于数学逻辑。图灵定理告诉我们，有些问题是不能解决的。布尔算术描述了计算机电路，数字分析在科学计算机中扮演着重要的角色。科学学科在尝试理解它们的系统是如何运作的。自然科学的存在是为了“填写上帝忘记给我们的说明书”。[11] 因此，在构建和测试自然现象的模型时，计算学属于科学学科。在设计和构建越来越大的计算系统时，我们采用的则是工程学的技术。

1989年，一组计算机科学教育家提出了一种课程模式，涵盖了计算历史中出现的三个方面的所有分区，这三个方面即理论（数学）、被计算机科学家称为抽象的实验（科学）和设计（工程学）[12]。理论指为理解一个领域中的对象之间的关系而构建的基本概念和符号。实验（抽象）指研究不同应用领域内的系统和体系结构的模型，判断这些模型是否预测了新的行为。设计指构造支持不同应用领域内的工作的计算机系统。

下表显示了这个特别工作组提出的分区大纲。

计算机科学的分区	计算机科学的分区	计算机科学的分区
算法和数据结构 程序设计语言 体系结构 数值和符号计算	操作系统 软件方法学和工程学 数据库和信息检索 人工智能和机器人技术	人机交互 图形学 组织信息学 生物信息学

在这12个分区中，有6个与理解和构建计算工具相关，它们是算法和数据结构、程序设计语言、（计算机）体系结构、操作系统、软件方法学和工程学以及人机交互。毫无疑问，它们被称为*系统分区*。另外6个分区则与计算机作为工具的用途相关，它们是数值和符号计算、数据库和信息检索、人工智能和机器人技术、图形学、组织信息学及生物信息学。这些分区被称为*应用分区*。

对系统分区和应用分区的研究都正在进行中。系统研究带来了更好的通用工具，应用研究为领域特定的应用提供了更好的工具。毫无疑问，把计算主题直接作为学科研究的人将影响那些把计算机用作工具的人。计算研究促成了人们日常使用的应用，而技术转变的速度也惊人地快。这种共生关系在计算学中比在其他学科中更强。

本书以绪论的方式解释了计算学科的基本思想。尽管本书有助于你成为更好的计算机用户，但这不是它存在的目的。我们的意图在于让你全面地了解计算机系统是如何工作的、它们现在处于什么阶段以及将来会向什么方向发展这些基础知识。因此，我们既要分析系统，也要分析应用。

小结

计算机系统由构成设备的硬件、机器执行的软件程序及由前两者管理和操作的数据组成，本书对计算机系统进行了概括性的研究。计算系统可以分为多个层次，本书将按照从内到外的顺序逐一介绍这些分层。

计算的历史使我们了解了现代计算系统的来源。计算的历史被划分为四个时代，每个时代都以用于构建硬件的元件和为了让用户更好地利用这些硬件而开发的软件工具为特征。这些工具构成了包围硬件的软件层。

在本书剩余的部分中，我们将分析构成计算系统的各个分层，从信息分层开始，到通信分层结束。我们的目标是让你理解和欣赏计算系统的方方面面。

你可能会继续对计算机科学做深度的研究，为计算系统的将来做出贡献。你也可能把计算机作为工具，成为其他学科的应用专家。无论你拥有什么样的未来，只有计算系统仍然盛行，对它们是如何运作的有一个基本了解都是必要的。

道德问题：数字化分裂

在过去几年中，社会对计算机技术的依赖程度在显著提高。使用电子邮件通信和访问Internet成了许多美国人日常生活中必不可少的一部分。美国商业部发布的统计信息显示，2000年有半数美国家庭都访问Internet。这意味着另一半美国家庭不能访问Internet或缺少使用它的技术。根据2004年Nielsen/NetRating调查问卷的结果，2.043亿的美国人在访问Internet。这个数字比美国总人口的三分之二还多。报告还发现，年龄在35～54岁之间访问Internet的妇女有81.7%，而同样年龄组的男士只有80.2%。年龄在25～34岁这组妇女有77%，而同组的男士只有75%。

大量的近期报告显示加拿大也遵循类似的百分比分布。此外，至少76%具有Internet连接的加拿大家庭现在具有宽带连接能力。美国稍弱一些，有60%的家庭具有宽带连接能力。许多美国家庭仍然不能访问Internet、不能高速连接和（或）没有使用Internet的技能。术语“数字化分裂”用于表示这种信息化时代“有”与“没有”之间的差距。

这种差距是日益增长的社会问题。农村人口、少数民族家庭、低收入家庭和残疾人士不能像那些具有优势的人群一样访问Internet。在教育方面，不同种族区的学校的计算机数量和质量以及联网能力都有很大差别。此外，只有硬件是不够的，教师也要接受使用这一技术的培训，理解它可以增加学生的知识。1996年建立的E-Rate计划是美国联邦政府支持的，这种计划针对学校和图书馆内的这些不平等之处，为贫困学校提供财政补贴。

在全球范围内看来，数字化分裂表明了发展中国家在向国际社会前进的过程中必须面对的又一个挑战。缺乏支持Internet访问必需的电信基础设施，新兴国家将处于非常不利的地位。世界上16%的人口使用了90%的Internet主机，正是存在这种差距的明显证据。事实上，整个非洲大陆的Internet连接比一个纽约市的还少。国际组织把这种国家间的科技差距列为最优先考虑的事。千禧年之初，创建了Digital Opportunity Task特别工作组（DOT特别工作组），这是在全球范围内扩展计算机技术的工作的开端。同样地，在2001年联合国建立了Information and Communications Technology（ICT）特别工作组，以解决国家间的数字化分裂问题。

数字化分裂暴露了计算机技术在美国国内和全球范围内对社会的严重冲击。毫无疑问，它是世界各国在整个21世纪和下个世纪中要继续解决的问题。

练习

从下列人名中选择练习1～10的答案。

A. Leibniz　　B. Pascal
C. Babbage　　D. Lovelace
E. Hollerith　　F. Byron
G. Turing　　H. Jacquard

1. 哪位法国数学家制造并出售了第一台齿轮传动的、能够计算加法和减法的机器？
2. 谁制造了第一台能够计算加法、减法、乘法和除法的机器？
3. 谁设计了第一台具有存储器的机器？
4. 谁是第一位程序设计员？
5. 谁提出了用穿孔卡片进行人口普查？
6. 谁编辑了Babbage的著作？
7. Ada Lovelace的父亲是谁？

8.《Code Breakers》这本书中提到了谁？

9. 谁提出了用于织布的孔的概念？

10. 谁与IBM相关？

在下列清单中，为练习11～23的硬件选出匹配的一代。

A. 第一代　　B. 第二代

C. 第三代　　D. 第四代

E. 第五代

11. 电路板

12. 晶体管

13. 磁芯存储器

14. 卡片输入/输出

15. 并行计算

16. 磁鼓

17. 磁带驱动器

18. 集成电路

19. 个人计算机

20. 真空管

21. 大规模集成电路

22. 磁盘

23. 连网

在下列清单中，为练习24～38的软件或软件概念选出匹配的一代。

A. 第一代　　B. 第二代

C. 第三代　　D. 第四代

E. 第五代

24. 汇编器

25. FORTRAN

26. 操作系统

27. 结构化程序设计

28. 分时

29. HTML（用于Web）

30. 装入器

31. 电子制表软件

32. 文字处理软件

33. Lisp

34. PC-DOS

35. 绑定在操作系统中的装入器和连接器

36. Java

37. SPSS

38. C++

为练习39～55中的问题提供简短的答案。

39. 如何理解20世纪80年代和90年代的特征是用户概述的改变这句话？

40. 请区分计算工具和计算学科。

41. 计算学属于数学学科、科学学科还是工程学学科？请解释。

42. 请区分计算学科中的系统分区和应用分区。

43. 请利用图1-2定义术语*抽象*。

44. 计算机科学中的系统分区与理解和构建计算工具相关，请列出这6种分区。

45. 计算机科学中的应用分区与计算机作为工具的用途相关，请列出这6种分区。

46. 请定义术语*协议*，并解释在计算中如何使用它。

47. 请区分机器语言和汇编语言。

48. 请区分汇编语言和高级语言。

49. FORTRAN和COBOL是在第二代计算机软件中定义的高级语言。比较这两种语言的历史和用途。

50. 请区分汇编器和编译器。

51. 请区分系统程序员和应用程序员。

52. 操作系统的基本原理是什么？

53. 什么构成了系统软件？

54. 下列软件的用途是什么？

  a) 装入器

  b) 连接器

  c) 编辑器

55. SPSS与它之前的程序有什么区别？

思考题

1. 请识别学校环境中的5种抽象，说明它们隐藏了什么细节，以及抽象如何有助于管理复杂度。

2. 请讨论*抽象*在计算机软件史上的角色。

3. 在你成长的过程中，家里有一台计算机吗？如果有，计算机是如何影响你的教育的？如果没有，那么缺少计算机对你的教育又有什么影响？

4. 数字化分裂把能够访问Internet的人和不能访问Internet的人分置在两边。你认为人人都应该能够访问Internet吗？
5. 解除数字化分裂需要花费大量金钱。你认为谁应该支付这笔费用？
6. 仅仅具有技术是不够的，人们还必须学习如何使用这些技术。你如何定义下列人群的计算机使用能力：
a) 工业化国家的中学生
b) 工业化国家的幼儿园教师
c) 工业化国家的大学生
d) 撒哈拉沙漠以南非洲的学生
e) 撒哈拉沙漠以南非洲的大学生
f) 安第斯山脉的政府公务员

第二部分 信息层

第2章 二进制数值和记数系统

我们在第1章中介绍了一些历史常用术语，现在可以真正地开始探讨这些计算技术了。这一章记述了计算机硬件用来表示和管理信息的方式——二进制数值。此外，这一章还把二进制数值置于各种记数系统中，帮助我们回忆这些初中学过的概念。虽然你可能已经知道了很多关于二进制数的概念，但是你也许从来没有意识到自己知道这些。所有记数系统的规则都一样，我们只不过是回顾那些基本概念，把它们应用到新的基数上。理解了二进制数值，就为理解计算系统如何使用二进制记数系统实现它们的任务做好了准备。

目标

学完本章之后，你应该能够：

- 区分数字分类
- 描述位置记数法
- 把其他基数的数字转换成十进制数
- 把十进制数转换成其他基数的数字
- 描述基数2、8和16之间的关系
- 解释以2的幂为基数的计算的重要性

2.1 数字和计算

数字对计算至关重要。除了使用计算机执行数字运算以外，所有使用计算机存储和管理的信息类型最终都是以数字形式存储的。在计算机的最底层，所有信息都只是用数字0和1存储的。因此，在开始研究计算机之前，首先需要探讨一下数字。

首先，回忆一下数字的分类，有自然数、负数、有理数、无理数等，它们在数学上很重要，但对理解计算却没有什么用。下面简短浏览一下相关的分类定义。

先定义一个总括的概念——数字。**数字**是属于抽象数学系统的一个单位，服从特定的顺序法则、加法法则和乘法法则。也就是说，数字表示一个值，可以对这些值施加某些算术运算。

现在，我们对数字进行分类。**自然数**是0和通过在0上重复加1得到的任何数，用于计数。**负数**是小于0的数，在相应的正数前加上负号即为负数。**整数**是所有自然数和它们的负数。**有理数**包括整数和两个整数的商，也就是说，任何有理数都可以被表示为一个分数。

> **数字**（number）：抽象数学系统的一个单位，服从算术法则。
>
> **自然数**（natural number）：0或通过在0上重复加1得到的任何数。
>
> **负数**（negative number）：小于0的数，是在相应的正数前加上负号。

> **整数**（integer）：自然数、自然数的负数或0。
> **有理数**（rational number）：整数或两个整数的商（不包括被0除的情况）。

这一章的重点是自然数以及在各种记数系统中如何表示它们。在讨论中，我们介绍了所有记数系统之间的关系。在第3章中，我们将分析负数和有理数的计算机表示法以及如何用数字表示其他形式的数据，如字符和图像。

本章中的部分资料可能是你所熟悉的。当然，一些基本概念是你应该知道的。一些基本的记数和运算规则是你经常使用的，所以你可能已经掌握了。本章的目标之一是让你回忆起这些基本规则，向你展示它们是如何应用到各个记数系统中的。这样，计算机使用二进制数值1和0来表示信息的思想就不那么难理解了。

2.2 位置记数法

943这个数中有多少实体？也就是说，943这个数表示多少件实物？用初中术语来说，943是9个100加4个10加3个1，或者说，是900个1加40个1加3个1。那么，754中又有多少实体？700个1加50个1加4个1。对吗？也许正确，答案是由你使用的记数系统的基数决定的。如果这些数字是以10为基数的，或者说是十进制数，也就是人们日常使用的数制，那么上述答案是正确的。但在其他记数系统中，上述答案就错了。

记数系统的**基数**规定了这个系统中使用的数字量。这些数字都是从0开始，到比基数小1的数字结束。例如，在以2为基数的系统中，有两个数字0和1。在以8为基数的系统中，有8个数字，从0到7。在以10为基数的系统中，有10个数字，从0到9。基数还决定了数位位置的含义。当给记数系统中的最后一个数加1后，必须执行数位位置左移。

> **基数**（base）：记数系统的基本数值，规定了这个系统中使用的数字量和数位位置的值。

数字是用**位置记数法**编写的。最右边的数位表示它的值乘以基数的0次幂，紧挨着这个数位的左边的数位表示它的值乘以基数的1次幂，接下来的数位表示它的值乘以基数的2次幂，再接下来的数位表示它的值乘以基数的3次幂，依此类推。也许你不知道自己对位置记数法如此熟悉。我们用它来计算943中的1的个数。

$$
\begin{aligned}
9 * 10^2 &= 9 * 100 = 900 \\
+4 * 10^1 &= 4 * \ \ 10 = \ \ 40 \\
+3 * 10^0 &= 3 * \ \ \ 1 = \ \ \ \underline{\ \ 3} \\
& \qquad\qquad\quad 943
\end{aligned}
$$

> **位置记数法**（positional notation）：一种表达数字的系统，数位按顺序排列，每个数位有一个位值，数字的值是每个数位和位值的乘积之和。[1]

位置记数法更正式的定义是用记数系统的基数的多项式表示的值。但什么是多项式呢？多项式是两个或多个代数项的和，每个代数项由一个常量乘以一个或多个变量的非负整数幂构成。在定义位置记数法时，变量指记数系统的基数。943可以表示为下列多项式，其中x表示基数。

$$9 * x^2 + 4 * x^1 + 3 * x^0$$

让我们来正式表述这一概念。如果一个数字采用的是以R为基数的记数系统，具有n个数

位，那么可以用下列多项式表示它，其中，d_i表示数字中第i位的数值。

$$d_n * R^{n-1} + d_{n-1} * R^{n-2} + \cdots + d_2 * R + d_1$$

是不是看起来很复杂？让我们看一个实例，以10为基数的数字63578。n等于5（该数字有5个数位），R等于10（基数）。根据公式，第5个数位（最左边的数位）乘以基数的4次方，第4个数位乘以基数的3次方，第3个数位乘以基数的2次方，第2个数位乘以基数的1次方，第一个数位什么都不乘。

$$6 * 10^4 + 3 * 10^3 + 5 * 10^2 + 7 * 10^1 + 8$$

在前面的计算中，我们都假设基数是10。这是一种逻辑假设，因为我们的记数系统是以10为基数的。但是，这并非意味着943表示的不会是一个以13为基数的值。如果是这样，要确定1的个数，必须先把943转换成以10为基数的数字。

$$\begin{array}{rl} 9 * 13^2 = 9 * 169 = & 1521 \\ +4 * 13^1 = 4 * \ \ 13 = & 52 \\ +3 * 13^0 = 3 * \ \ \ 1 = & \underline{\ \ \ 3} \\ & 1576 \end{array}$$

因此，以13为基数的数943等于以10为基数的数1576。记住，这两个数是等值的。也就是说，它们表示的是同等数量的实体。如果一个包中有（以13为基数）943个豆子，另一个包中有（以10为基数）1576个豆子，那么两个包中的豆子数是完全一样的。记数系统使我们能用多种方式表示数值。

注意，以10为基数，最右边的数字是“1”数位；以13为基数，最右边的数字也是“1”数位。事实上，以任何数字为基数，最右边的数字都是“1”数位，因为任何数字的0次幂都是1。

为什么有人要把数值表示为以13为基数的呢？虽然以13为基数的数并不常见，但是有时它对理解记数系统的运作还是很有帮助的。例如，有一种计算技术称为*散列法*，就是将数字打乱，方法之一就是用另一种基数表示这个数字。

其他基数（如2）在计算机处理中更加重要。我们来详细探讨一下这些基数。

0的重要性

位置记数法之所以能存在，完全是因为0这个概念。我们通常认为，0是所有现代数学分支的交集中的基本概念。Georges Ifrah在他的著作《Universal History of Computing》中说道：“总而言之，0的发现给了人类思想无限的潜力。没有其他的人类创新可以给人类智能的发展带来如此深远的影响”。[2]

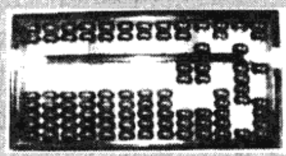
算　盘

在第1章的计算简史中，我们提到了算盘这种早期的计算设备。更确切地说，算盘是使用位置记数法表示十进制数的设备。每一列中的算珠表示那一列的数字。所有列组合在一起表示一个完整的数字。



Theresa DiDonato提供的图片

中部横木以上的算珠表示5个单位，以下的算珠表示1个单位。没有挨着中部横木的算珠与得到的数字无关。下面的图显示了用算盘表示的数字27 091。


Theresa DiDonato提供的图片

用户通过以特定的方式移动算珠来执行计算，反应了基本的算术运算，即加法、减法、乘法和除法。

尽管很古老，但是现在在许多亚洲文化中还是能见到算盘。在商店里，收银员使用的可能是算盘，而不是电子收银机。虽然没有电子设备的优点，算盘能更有效地满足基本商务需要的计算。算盘能手能在速度和正确度上与使用计算器的用户一比高下。

亚洲国家的孩子会学习算盘的机械化操作，和你反复背诵乘法表非常相似。要对一个数字执行运算，用户只需要用一只手的拇指、食指和中指执行一系列算珠移动即可。这些移动对应于单独的数位，由执行的运算决定。例如，算盘上已有数字5，要把7加到5上，用户需要清除表示5的算珠（把它移到算盘顶部），把这一列中下面的两个算珠推到横木处，在这一列左边的列中推上一个算珠。虽然这些移动操作与我们在纸上所做的基本加法运算一样，但是算盘的用户并没有考虑数学，他们习惯于在特定的数字遇到特定的运算时，执行特定的移动操作。当计算完成后，用户将读取算盘上显示的结果。

双五进制表示法

IBM 650控制台是20世纪50年代末期流行的商用计算机，它允许运算符读取使用双五进制系统的内存的内容。这种数字表示系统使用7个灯表示10个十进制数。


IBM Corporate Archives提供的图片

每个数字由两个灯表示，一个灯属于上部的两个灯，另一个灯属于下部的5个灯。如果左上部的灯亮了，其他5个灯从上到下分别表示0、1、2、3和4。如果右上部的灯亮了，其他5个灯从上到下分别表示5、6、7、8和9。下图表示的是数字7。



IBM 650被称为计算机的Ford Tri-Motor，因为像Ford Tri-Motor一样，IBM 650被运载到拉丁美洲，在那里延长了它的寿命。

2.2.1 二进制、八进制和十六进制

以2为基数（二进制）的记数系统在计算中尤其重要。了解以2的幂为基数的记数系统（如以8为基数的八进制和以16为基数的十六进制）也很有用。记住，基数规定了记数系统中数字的个数。以10为基数的记数系统具有10个数字（0－9），以2为基数的记数系统具有2个数字（0－1），以8为基数的记数系统具有8个数字（0－7）。因此，数字943不可能表示一个基数小于10的值，在这样的记数系统中，根本不存在数字9。它是一个以10或大于10的数为基数的有效数字。同样地，2074是一个以8或大于8的数为基数的有效数字，不存在于以小于8的数字为基数的记数系统中（因为它使用了7）。

那么在基数大于10的记数系统中有哪些数字呢？我们用符号表示相当于十进制中大于等于10的值的数字。在以比10大的数为基数的记数系统中，我们把字母用作数字。字母A表示数字10，字母B表示11，C表示12，依此类推。因此，以16为基数的记数系统中的16个数字如下所示：

0、1、2、3、4、5、6、7、8、9、A、B、C、D、E和F

让我们看一些八进制、十六进制和二进制的数，看看它们表示的十进制数是什么。例如，让我们计算与八进制数（以8为基数）754等值的十进制数。如前所示，我们把这个数字展开成多项式的形式，然后求和。

$$
\begin{aligned}
7 * 8^2 &= 7 * 64 = 448 \\
+5 * 8^1 &= 5 * \ \ 8 = \ \ 40 \\
+4 * 8^0 &= 4 * \ \ 1 = \ \ \underline{\ \ 4} \\
& \qquad\qquad\quad 492
\end{aligned}
$$

把十六进制数ABC转换成十进制数：

$$
\begin{aligned}
A * 16^2 &= 10 * 256 = 2560 \\
+B * 16^1 &= 11 * \ \ 16 = \ \ 176 \\
+C * 16^0 &= 12 * \ \ \ 1 = \ \ \underline{\ \ 12} \\
& \qquad\qquad\qquad 2748
\end{aligned}
$$

注意，我们把数字转换成十进制数所执行的操作完全一样，只不过这次使用的基数是16，我们必须记住字母数字表示的数值。多加练习，你就不会觉得把字母用作数字很奇怪了。

最后，我们来把二进制（以2为基数的）数1010110转换成十进制数，执行的步骤仍然相同，只是基数改变了：

$$
\begin{aligned}
1 * 2^6 &= 1 * 64 = 64 \\
+0 * 2^5 &= 0 * 32 = \ \ 0 \\
+1 * 2^4 &= 1 * 16 = 16 \\
+0 * 2^3 &= 0 * \ \ 8 = \ \ 0 \\
+1 * 2^2 &= 1 * \ \ 4 = \ \ 4 \\
+1 * 2^1 &= 1 * \ \ 2 = \ \ 2 \\
+0 * 2^0 &= 0 * \ \ 1 = \ \ \underline{\ \ 0} \\
& \qquad\qquad\quad 86
\end{aligned}
$$

还记得吗？任何记数系统中的最大数字比基数小1。要用任何基数表示基数值，只需要两个数字。0位于最右边，1在0的左边，这个数字表示基数值本身。因此，10是以10为基数的记数系统中的10，10是以8为基数的记数系统中的8，10是以16为基数的记数系统中的16。仔细考虑一下。记数系统的一致性是非常好的。

采用其他基数的数字的加法和减法运算与十进制数中的运算完全一样。

2.2.2 其他记数系统中的运算

回忆一下十进制数运算的基本思想。0+1等于1，1+1等于2，2+1等于3，依此类推。当要相加的两个数的和大于基数时，情况就变得比较有趣了。例如，1+9。因为没有表示10的符号，所以只能重复使用已有的数字，并且利用它们的位置。最右边的值将回0，它左边的位置上发生进位。因此，在以10为基数的记数系统中，1+9等于10。

二进制运算的规则与十进制运算的类似，不过可用的数字更少。0+1等于1，1+1等于0加一个进位。同样的规则适用于较大数中的每一个数位，这一操作将持续到没有需要相加的数字为止。下面的例子将求二进制数101110和11011的和。每个数位之上的值标识了进位。

```
 11111      ←进位
  101110
+  11011
 -------
 1001001
```

可以通过把两个运算数都转换成十进制数，用它们的和与上面的值比较来确认这个答案是否正确。101110等于十进制的46，11011等于27，它们的和是73。1001001等于十进制的73。

在初中学过的减法法则是9－1等于8，8－1等于7，依此类推，直到要用一个较小的数减一个较大的数，例如0－1。要实现这样的减法，必须从减数数字中的下一个左边数位上“借1”。更确切地说，借的是基数的一次幂。因此，在十进制中，借位时借到的是10。同样的逻辑适用于二进制减法。在二进制减法中，每次借位借到的是2。下面的两个例子中标识出了借位。

```
    1     ←借位
   022
 111001
-   110
 ------
 110011
```

同样地，可以通过把所有值转换成十进制的，进行减法运算后与上面的结果进行比较，看答案是不是正确的。

2.2.3 以2的幂为基数的记数系统

二进制数和八进制数有种非常特殊的关系。给定一个二进制数，可以很快读出它对应的八进制数，给定一个八进制数，也可以很快读出它对应的二进制数。以八进制数754为例，如果把每个数位都替换成这个数位的二进制表示，就可以得到754对应的二进制数。也就是说，八进制中的7等于二进制的111，八进制的5等于二进制的101，八进制的4等于二进制的100，所以八进制的754等于二进制的111101100。

为了便于转换，下表列出了从0到10的十进制数和它们对应的二进制数及八进制数。

二进制	八进制	十进制	二进制	八进制	十进制
0	0	0	110	6	6
1	1	1	111	7	7
10	2	2	1000	10	8
11	3	3	1001	11	9
100	4	4	1010	12	10
101	5	5			

把二进制数转换成八进制数，要从最右边的二进制数位开始，每三个数位一组，把每组数字转换成相应的八进制数。

$$\frac{111}{7}\quad\frac{101}{5}\quad\frac{100}{4}$$

下面把二进制数1010110转换成八进制的，然后把这个八进制数转换成十进制的。答案应该是1010110对应的十进制数86。

$$\frac{1}{1}\quad\frac{010}{2}\quad\frac{110}{6}$$

$$\begin{aligned} 1 * 8^2 &= 1 * 64 = 64 \\ +2 * 8^1 &= 2 * \ \ 8 = 16 \\ +6 * 8^0 &= 6 * \ \ 1 = \underline{\ \ 6} \\ &\qquad\qquad 86 \end{aligned}$$

二进制数和八进制数之间可以快速转换的原因在于8是2的幂。在二进制和十六进制之间也存在类似的关系。让我们把二进制数1010110转换成十六进制的，从右到左，把每四个数位分成一组。

$$\frac{101}{5}\quad\frac{0110}{6}$$

$$\begin{aligned} 5 * 16^1 &= 5 * 16 = 80 \\ +6 * 16^0 &= 6 * \ \ 1 = \underline{\ \ 6} \\ &\qquad\qquad 86 \end{aligned}$$

现在，我们来把十六进制数ABC转换成二进制的。表示一位十六进制数需要四位二进制数。十六进制中的A等于十进制中的10，因此，等于二进制的1010。同样地，十六进制的B等于二进制的1011，十六进制的C等于二进制的1100。因此，十六进制数ABC等于二进制的101010111100。

我们不直接把10001001010转换成十进制的2748，而是把它划分成八进制数位，转换成八进制数。

$$\frac{101}{5}\quad\frac{010}{2}\quad\frac{111}{7}\quad\frac{100}{4}$$

八进制的5274等于十进制的2748。

在下一节中，我们将说明如何把十进制数转换成其他记数系统中的等值数字。

可以数到三吗？

认知心理学家已经证明学龄前儿童所能识别的集合不超过三个，即一个对象的集合、两个对象的集合和三个或多于三个（又称为多个）对象的集合。人类学家和语言学家也确认了两个多世纪前，许多语言都只有两个或三个表示数字的单词，即“一个”、“一对”和“许多”。英语中仍有一些反映三个或多个的单词，如“gang”、“pile”、“bunch”、“flock”、“herd”、“school”、“fleet”、“pride”、“pack”和“gaggle”。

——Denise Schmandt-Besseerat, *One, Two…Three, Odyssey*, September/October 2002, p6和7

2.2.4 把十进制数转换成其他数制的数

转换十进制数的规则涉及新基数的除法。由这个除法可以得到一个商和一个余数。余数将成为新数字中的（从右到左）下一位数，商将代替要转换的数字。这一过程将持续到商为0为止。让我们用另一种形式来描述这些规则。

```
While（商不是0）
  用新基数除这个十进制数
  把余数作为答案左边的下一个数字
  用商代替这个十进制数
```

这些规则构成了把十进制数转换成其他数制的*算法*。算法是解决问题的步骤的逻辑序列。后面的章节中将有大量关于算法的介绍。这里我们只是介绍一种描述算法的方式，说明如何用它来执行转换。

算法的第一行告诉我们，在除法的商成为0之前，要重复执行下面的三行操作。让我们来把十进制数2748转换成十六进制数。我们在前面的例子中看到了，答案应该是ABC。

```
     171    ←商
 16√2748
    16
    114
    112
      28
      16
      12    ←余数
```

余数（12）是十六进制数中的第一位数，由数字C表示。迄今为止，答案是C。由于商不是0，所以要用新基数除它（171）。

```
     10    ←商
 16√171
    16
     11    ←余数
```

余数（11）是答案中左边的下一位数，由数字B表示。现在，答案是BC。由于商不是0，所以要用新基数除它（10）。

$$\begin{array}{rl} 0 & \leftarrow 商 \\ 16\sqrt{10} & \\ \underline{0} & \\ 10 & \leftarrow 余数 \end{array}$$

余数（10）是答案中左边的下一位数，由数字A表示。现在，答案是ABC。由于商是0，所以整个过程结束了，最后的答案是ABC。

2.2.5 二进制数值和计算机

虽然有些早期计算机是十进制机器，但是现代计算机都是二进制机器。也就是说，计算机中的数字都是用二进制形式表示的。事实上，所有信息都是用二进制数值表示的，原因在于计算机中的每个存储位只有高电压和低电压两种信号。由于每个存储位的状态只能是这两者之一，所以用0和1表示这两种状态很符合逻辑。低电压信号等同于0，高电压信号等同于1。事实上，你可以忘记电压，认为每个存储位存放的值是0或1。注意，存储位不能是空的，必须存放0或1。

每个存储单元称为一个**二进制数字**（或简称为**位**）。把位集合在一起就构成了**字节**（8位），字节集合在一起构成了**字**。字中的位数称为计算机的字长。例如，20世纪70年代晚期的IBM 370体系结构中有半字（2字节或16位）、全字（4字节）和双字（8字节）。

> **二进制数字**（binary digit）：二进制记数系统中的一位数字，可以是0或1。
> **位**（bit）：二进制数字的简称。
> **字节**（byte）：8个二进制位。
> **字**（word）：一个或多个字节，字中的位数称为计算机的字长。

现代计算机通常是32位的机器（如Intel公司的Pentium IV处理器）或64位的机器（如HP公司的Alpha处理器和Intel公司的Itanium 2处理器）。但是有些应用设备（如寻呼机）使用的微处理器是8位的机器。无论你在使用的是什么计算机，它们最终采用的都是二进制记数系统。

关于计算机和二进制数之间的关系，还有很多是值得探讨的。在下一章中，我们将分析各种类型的数据，看看它们在计算机中是如何表示的。在第4章中，我们将介绍如何控制表示二进制数值的电信号。第7章将介绍如何用二进制数表示计算机执行的程序命令。

Grace Murray Hopper



美国海军少将Grace Murray Hopper生于1943年，于1992年元旦去世，她的生活与计算密不可分。1991年，由于她在计算机程序设计语言开发方面的杰出贡献，被授予美国国家科技奖章。这一贡献简化了计算技术，为广大用户打开了一扇大门。

海军少将Hopper于1906年12月9日出生在纽约市的Grace Brewster Murray家。她曾就读于Vassar大学，并从耶鲁大学获得了数学博士学

位。之后的10年中，她在Vassar大学教授数学。

1943年，Hopper加入了美国海军，被分配到哈佛大学的军械计算项目处，担任Mark I的程序员。战争过后，她仍留在哈佛大学，担任教员，继续从事有关海军的Mark II和Mark III计算机的工作。1949年，她加入了Eckert-Mauchly Computer公司，从事有关UNIVAC I的工作。就是在这里，她对计算做出了富有传奇色彩的贡献，发现了第一个计算机bug，即计算机硬件中的一个问题。

1952年，Hopper得到了一台能够运行的编译器，当时普遍认为计算机只能进行算术运算。虽然Hopper并不属于设计计算机语言COBOL的委员会，但她也积极参与了这种语言的设计、实现和使用。COBOL（面向商业的通用语言，Common Business-Oriented Language）是在20世纪60年代早期开发的，目前仍广泛应用于商业数据处理中。

1966年，Hopper从海军退休了，但同年即被招回，负责指导海军维护程序设计语言的一致性。就像海军将领Hyman Rickover被称为美国有核海军之父一样，海军少将Hopper是美国海军计算机化的数据自动化之母。直到1986年再次退休，她一直服务于海军数据自动化司令部，军衔为海军少将。在去世时，她是Digital Equipment公司的高级顾问。

在Hopper的一生中，曾收到过来自40多所学院和大学的荣誉学位。此外，她还获得过多种奖励，包括Data Processing Management Association授予的第一个计算机科学年度人物奖，和Special Interest Group for Computer Science Education（隶属于ACM，Association for Computing Machinery）授予的计算机科学教育贡献奖。

Hopper喜爱年轻人，喜欢在学院和大学校园中进行讲座。她常常分发彩色的电线，她称之为“一毫微秒”，因为这些电线的长度是一英尺，即光速运行一毫微秒（十亿分之一秒）的距离。她对年轻人的教诲是“你掌管着一切，领导着人们。我们要极度热衷于管理，而忘记自己的领导身份。”

当被问道在她的诸多成就中她最为自豪的是什么时，她答道“我多年来培养出的所有年轻人。”

小结

数是用位置记数法编写的，其中数字按顺序排列，每个数字具有一个位值，数值等于每个数字与它的位值的乘积之和。位值是记数系统的基数的幂。因此，在十进制记数系统中，位值是10的幂；在二进制记数系统中，位值是2的幂。

任何用位置记数法表示的数都可以进行算术运算。十进制数的运算规则也适用于其他记数系统。给记数系统中的最大数字加1将引发进位。

二进制数、八进制数和十六进制数是相关的，因为它们的基数都是2的幂。这种关系为它们之间的数值转换提供了快捷方式。计算机硬件采用的是二进制数。低电压信号相当于0，高电压信号相当于1。

道德问题：计算机和国家安全

无正当理由不能搜查人身、房屋、文件以及私人财产，这些都不容冒犯。不应下达这样的搜查证，但是一些特殊情况下，如果有法庭的支持，并且清楚说明了要搜查的地点和要逮捕的人或要查封的物品，那么还是可以搜查的。

当FBI宣布要对Internet进行监控的计划时，个人隐私权组织和政客们都被激怒了。但经历了2001年的911袭击后，新美国爱国者法案为FBI的计划提供了支持。Internet服务提供商（ISP）那里被安装了一种名为Carnivore的工具，它能够扫描并收集所有经过ISP主机的数据。包括恐怖主义者在内的各种犯罪分子已经使用了多年的高科技来计划和执行非法活动。电子邮件、网络站点、银行和电话线都被用到这些活动中，经证明，Internet是策划这些破坏活动的有力工具。

根据FBI的结论，执法部门必须要有在电脑空间中跟踪犯罪线索的能力。在纽约和宾夕法尼亚州遭袭击之前，部署了一些Carnivore代理，这次代理至少暂时阻止了一些恐怖袭击。美国爱国者法案和国家安全法案的通过，给FBI捕捉犯罪分子和恐怖分子开了绿灯。政府和执法部门指出，有了电子邮件和网络，犯罪分子就能够不受时间和空间的限制，召集成员、制订计划以及进行沟通。如果没有Carnivore这样的侵入技术，执法人员就不能跟踪并证明这些非法活动。

Carnivore的支持者还指出，该软件只用于那些受怀疑的目标，所以虽然所有经过ISP的数据都会被收集起来，但是提取的却是特定的内容。此外，FBI还要向联邦或州的司法部长申请这一工具的使用权，声明要收集哪些数据以及收集谁的数据。他们还要提供官方证明，说明其他监视方式都失败了，或者太危险了。这种工具规定的用途是搜集证据，而不是情报。

个人隐私权拥护者对政府关于在哪里使用Carnivore的声明并不满意，适当的约束（如前面所提到的）可以防止Carnivore滥用的保证也不能让他们释怀。他们担心，使用这种技术为其他侵犯隐私权的做法打开了一道大门。例如，Carnivore能够跟踪某个特定ISP的所有客户的网上冲浪爱好。它所能跟踪的不仅仅是电子邮件，还有即时消息、在线购物以及其他任何通过ISP的信息。对于个人隐私权组织来说，这无疑冒犯了第四修正案。除了宪法问题外，还存在由不道德的人滥用、错用或恶意使用的风险。任何具有ISP上的Carnivore系统访问密码的人都能访问这个ISP接收和发送的所有数据，简而言之，这样的风险太大，远远超出了跟踪几条具有犯罪企图的数据的意图。

练习

为练习1～5选择与它们匹配的定义。

A. 数字　　B. 自然数

C. 整数　　D. 负数

E. 有理数

1. 抽象数学系统的一个单位，服从算术法则。
2. 自然数、自然数的负数或0。
3. 0或通过在0上重复加1得到的任何数。
4. 整数或两个整数的商（不包括被0除的情况）。
5. 小于0的数，是在相应的正数前加上负号。

为练习6～11选择与问题匹配的答案。

A. 10001100　　B. 10011110

C. 1101010　　D. 1100000

E. 1010001　　F. 1111000

6.1110011 + 11001（二进制加法）

7.1010101 + 10101（二进制加法）

8.1111111 + 11111（二进制加法）

9.1111111 − 111（二进制减法）

10.1100111 − 111（二进制减法）

11.1010110 − 101（二进制减法）

判断练习12～17是对是错：

A. 对　　B. 错

12. 二进制数在计算中很重要，因为二进制数可以被转换成以任何数为基数的数。
13. 可以迅速地读出一个二进制数对应的十六进制数，但是不能迅速地读出它对应的八进制数。
14. 从左到右，每四个二进制数字可以被转换成一个十六进制数字。
15. 一个字节由6个二进制数字构成。
16. 一个字节中不能存储两个十六进制数字。
17. 无论从左到右，还是从右到左，把一个八进制数转换成二进制数得到的结果相同。

练习18～45是问题或简答题。

18. 请区分自然数和负数。
19. 请区分自然数和有理数。

20. 把下列数标识为自然数、负数或有理数。
    a) 1.333333 b) −1/3
    c) 1066 d) 2/5
    e) 6.2 f) π(pi)
21. 采用下列基数时，891中有多少个1?
    a) 以10为基数 b) 以8为基数
    c) 以12为基数 d) 以13为基数
    e) 以16为基数
22. 用练习21中的各种基数，把891表示为多项式的形式。
23. 把下列数转换成十进制数。
    a) 111（以2为基数） b) 777（以8为基数）
    c) FEC（以16为基数） d) 777（以16为基数）
    e) 111（以8为基数）
24. 请解释基数2和基数8之间的关系。
25. 请解释基数8和基数16之间的关系。
26. 请扩展2.2.3节中的表，加入数11至16。
27. 请扩展练习26中的表，加上十六进制的数。
28. 请把下列二进制数转换成八进制的。
    a) 111110110 b) 1000001
    c) 10000010 d) 1100010
29. 请把下列二进制数转换成十六进制的。
    a) 10101001 b) 11100111
    c) 01101110 d) 01121111
30. 请把下列十六进制数转换成八进制的。
    a) A9 b) E7
    c) 6E
31. 请把下列八进制数转换成十六进制的。
    a) 777 b) 605
    c) 443 d) 521
    e) 1
32. 请把下列十进制数转换成八进制的。
    a) 901 b) 321
    c) 1492 d) 1066
    e) 2001
33. 请把下列十进制数转换成二进制的。
    a) 45 b) 69
    c) 1066 d) 99
    e) 1
34. 请把下列十进制数转换成十六进制的。
    a) 1066 b) 1939
    c) 1 d) 998
    e) 43
35. 如果你要表示十八进制记数系统中的数，那么除了字母外，用什么符号表示十进制数16和17?
36. 用你在练习35中设计的符号把下列十进制数转换成十八进制的。
    a) 1066 b) 99099
    c) 1
37. 执行下列八进制加法运算。
    a) 770 + 665 b) 101 + 707
    c) 202 + 667
38. 执行下列十六进制加法运算。
    a) 19AB6 + 43 b) AE9 + F
    c) 1066 + ABCD
39. 执行下列八进制减法运算。
    a) 1066 − 776 b) 1234 − 765
    c) 7766 − 5544
40. 执行下列十六进制减法运算。
    a) ABC − 111 b) 9988 − AB
    c) A9F8 − 1492
41. 为什么二进制数在计算学中很重要?
42. 一个字节包含多少位?
43. 在64位机器中，有多少字节?
44. 为什么像寻呼机这样的微处理器的字长只有8位?
45. 为什么学习如何操作定长数字很重要?

思考题

1. 练习20要求指出π所属的类别。π并不属于任何指定的分类，它和e是超越数。在字典或旧数学书中查找超越数，用你自己的话定义它。
2. 复数是另一类本章没有讨论的数。在字典或旧数学书中查找复数，用你自己的话定义它。
3. 许多每天发生的事都可以用二进制数位表示。

例如，门是打开的还是关闭的，炉子是开着的还是关着的，狗是睡着了还是醒着的。那么关系可以被表示为二进制值吗？请举例回答。

4. 应该允许政府官员使用像Carnivore这样的高科技技术监控可能威胁个人或国家安全的在线信息吗？为什么允许这样做？为什么不允许？

5. 以你的观点，像Carnivore这样的技术是否与第四修正案关于隐私权的权利冲突呢？或者它们是否是在911之后对抗美国或其他国家所面临的新威胁必不可少的手段呢？

第3章 数据表示法

在旅行时，你可能需要一张地图。地图并不是你游历的地点，而是这些地点的一种表示，它具有从一个地点到另一个地点所必需的信息。

同样地，我们需要一种方法来表示计算机存储和管理的数据，这种方法要能够捕捉信息的要素，而且必须采用便于计算机处理的形式。第2章介绍了二进制记数系统的基本概念，这一章将探讨如何表示和存储计算机管理的各种类型的数据。

目标

学完本章之后，你应该能够：

- 区分模拟数据和数字数据
- 解释数据压缩和计算压缩率
- 解释负数和浮点数的二进制格式
- 描述ASCII和Unicode字符集的特征
- 执行各种类型的文本压缩
- 解释声音的本质和它的表示法
- 解释RGB值如何定义颜色
- 区分光栅图形和矢量图形
- 解释时间和空间视频压缩

3.1 数据和计算机

没有数据，计算机就毫无用处。计算机执行的每个任务都是在以某种方式管理数据。因此，用适当的方式表示和组织数据是非常重要的。

首先，我们来区别一下术语**数据**和**信息**。虽然这两个术语通常可以互换使用，但分清它们还是有用的，尤其对计算来说更是如此。数据是基本值或事实，而信息则是用某种能够有效解决问题的方式组织或处理过的数据。数据是未组织过的，缺少上下文。信息则可以帮助我们回答问题（即"告知"）。当然，这种区别是相对于用户的需求而言的，但它却正是计算机在协助我们解决问题时所扮演的角色的本质。

> **数据**（data）：基本值或事实。
>
> **信息**（information）：用有效的方式组织或处理过的数据。

本章的重点是各种类型的数据的表示法。在后面的几章中，将讨论各种组织数据来解决特定类型的问题的方法。

不久以前，计算机处理的几乎都是数字和文本数据，但现在它已经成为真正的**多媒体**设备，可以处理各种各样的信息。计算机可以存储、表示和帮助我们修改各种类型的数据，包括：

- 数字

- 文本
- 音频
- 图像和图形
- 视频

这些数据最终都被存储为二进制数字。每个文档、图像和广播讲话都将被表示为由0和1组成的字符串。这一章将依次探讨每种数据类型，介绍在计算机上表示这些数据类型的方式的基本思想。

如果不讨论**数据压缩**，就不能讨论数据表示法。所谓数据压缩，就是减少存储一段数据所需的空间。过去，由于存储的局限性，我们需要使数据尽可能的小。现在，计算机存储变得比较便宜了，但是我们有更迫切的理由来缩短数据，因为我们要与其他人共享数据。网站和它底层的网络具有固有的**带宽**限制，定义了在固定时间内从一个地点传输到另一个地点的最大位数或字节数。

> **多媒体**（multimedia）：几种不同的媒体类型。
> **数据压缩**（data compression）：减少存储一段数据所需的空间。
> **带宽**（bandwidth）：在固定时间内从一个地点传输到另一个地点的最大位数或字节数。

压缩率说明了压缩的程度，是原始数据的大小除压缩后的数据大小。计算压缩率的值可以是位数、字符数或其他各种适用的单位，只要这两个值采用的单位相同即可。压缩率是一个0到1之间的数。压缩率越接近0，压缩程度越高。

数据压缩技术可以是**无损的**，即提取的数据没有丢失任何原始信息。数据压缩也可以是**有损的**，即在压缩过程中将丢失一些信息。尽管我们从来都不想丢失信息，但在某些情况下，损失是可以接受的。在处理数据表示法和压缩时，我们总要在精确度和数据大小之间做出权衡。

> **压缩率**（compression ratio）：原始数据的大小除压缩后的数据大小。
> **无损压缩**（lossless compression）：不会丢失信息的数据压缩技术。
> **有损压缩**（lossy compression）：会丢失信息的数据压缩技术。

3.1.1 模拟数据和数字数据

自然界的大部分都是连续的和无限的。实数直线图像是连续的，直线中的数值可以是无限大或无限小的。也就是说，给定任意的数，总可以找到比它大或比它小的数。两个整数之间的数字空间是无限的。例如，任何数都可以被均分。但是，世界并非只是数学意义上的无限。色谱是无限种色度的连续排列。现实世界中的对象在连续的无限空间中移动。理论上说来，可以给出你和墙之间的距离，但你却绝对无法真正到达那堵墙。

另一方面，计算机则是有限的。计算机内存和其他硬件设备用来存储和操作一定量的数据的空间只有那么多。用有限的机器表示无限的世界，我们从来都没有成功过。然而我们的目标是使表示的世界满足我们的计算需要和视觉及听觉官能。我们想使自己的表达法能够满足所有作业。

表示数据的方法有两种，即模拟法和数字法。**模拟数据**是一种连续表示法，模拟它表示的真实信息。**数字数据**是一种离散表示法，把信息分割成了独立的元素。

> **模拟数据**（analog data）：用连续形式表示的信息。
> **数字数据**（digital data）：用离散形式表示的信息。

水银温度计是一种模拟设备。水银柱按温度的正比例在管子中升高。我们校准这个管子，给它标上刻度，以便能够阅读当前的温度，通常是一个整数，如华氏75度。但是，水银温度计升温时实际采用的是连续的方式。有时，实际温度是华氏74.568度，水银柱的确指在相应的位置，但即使我们的标记再详细，也不足以反映出这么细微的改变。请参阅图3-1。

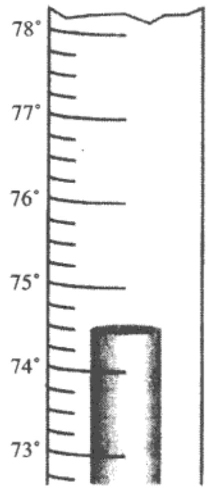


图3-1 按温度的正比例连续升高的水银温度计

模拟数据完全对应于我们周围连续无限的世界。因此，计算机不能很好地处理模拟数据。我们需要**数字化**数据，把信息分割成片段，单独表示每个片段。我们在这一章中讨论的每种表示法都是把一个连续的实体分割成离散的元素，然后用二进制数字单独表示每个离散元素。

> **数字化**（digitize）：把信息分割成离散的片段。

但为什么使用二进制呢？从第2章可以了解到，二进制只是众多等价的记数系统中的一员。那么能使用我们所熟悉的十进制吗？可以。事实上，采用十进制的计算机早就出现了。但是，现代计算机使用和管理的都是二进制数值，因为如果存储和管理数据的设备只需要表示两种数值之一，那么费用要小得多，而且也可靠得多。

此外，如果电信号只传输二进制数据，也易于维护。表示模拟信号的电压持续地上下波动，但是数字信号却只有高低两种状态，对应两个二进制数字。请参阅图3-2。



图3-2 模拟信号和数字信号

在沿线下降时，所有电信号（包括模拟信号和数字信号）都会降级。也就是说，由于环境影响，信号的电压会波动。问题是，当模拟信号降级时，信息就会丢失。由于任何电压都是有效的，所以不可能知道原始的信号状态，甚至不能知道该信号是否改变过。

另一方面，数字信号只在两个极端之间跳跃，被称为**脉冲编码调制**（PCM）。数字信号在信息丢失之前可以降级相当多，因为大于某个阈值的电压值都被看作高电压，小于这个阈值的电压值都被看作低电压。数字信号会被周期性地**重新计时**，恢复到它的原始状态。只要在信号降级太多之前重新计时，就不会丢失信息。图3-3展示了模拟信号和数字信号的降级效应。

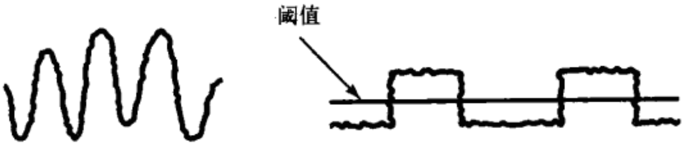


图3-3 模拟信号和数字信号的降级效应

> **脉冲编码调制**（pulse-code modulation）：电信号在两个极端之间跳跃的变化。
> **重新计时**（reclock）：在信号降级太多之前将它重置为原始状态的行为。

3.1.2 二进制表示法

在开始单独介绍各种数据类型的表示法之前，要记住二进制的固有特性。一个位只能是0

或1，没有其他的可能，因此，一个位只能表示两种状态之一。例如，如果我们要把食物分成甜的和酸的两类，那么只用一位二进制数字即可。可以规定，0表示食物是甜的，1表示食物是酸的。但是，如果要表示更多的分类（例如辣的），一位二进制数字就不能胜任了。

要表示多于两种的状态，需要多个位。两个位可以表示四种状态，因为两个位可以构成四种0和1的组合，即00、01、10和11。例如，如果要表示一辆汽车采用的是四种档（停车、发动、倒车和空档）中的哪一种，只需要两位二进制数字即可。停车由00表示，发动由01表示，倒车由10表示，空档由11表示。位组合与它们表示的状态之间的实际映射有时是无关的（如果你愿意，可以用00表示倒车档），然而有时这种映射是有意义的，很重要，我们将在本章后面的小节中讨论这一点。

如果要表示的状态多于四种，需要两个以上的位。三位二进制数字可以表示8种状态，因为三位数字可以构成8种0和1的组合。同样地，四位二进制数字可以表示16种状态，五位可以表示32种，依此类推。请参阅图3-4。注意，每列中的位组合都是二进制的。

1位	2位	3位	4位	5位
0	00	000	0000	00000
1	01	001	0001	00001
	10	010	0010	00010
	11	011	0011	00011
		100	0100	00100
		101	0101	00101
		110	0110	00110
		111	0111	00111
			1000	01000
			1001	01001
			1010	01010
			1011	01011
			1100	01100
			1101	01101
			1110	01110
			1111	01111
				10000
				10001
				10010
				10011
				10100
				10101
				10110
				10111
				11000
				11001
				11010
				11011
				11100
				11101
				11110
				11111

图3-4 位组合

一般说来，n位二进制数字能表示2^n种状态，因为n位数字可以构成2^n种0和1的组合。请注意，每当可用的位数增加一位，可以表示的状态的数量就会多一倍。

让我们把这个问题反过来。要表示25种状态，需要多少位？四位二进制数字是不够的，因为四位数字只能表示16种状态。至少需要五位二进制数字，它们可以表示32种状态。由于我们只需要表示25种状态，所以有些位组合没有有效的解释。

记住，即使技术上只需要最少的位数来表示一组状态，而我们可能会多分配一些位数。计算机体系结构一次能够寻址和移动的位数有一个最小值，通常是2的幂，如8、16或32。因此，分配给任何类型的数据的最小存储量通常是2的幂的倍数。

3.2 数字数据的表示法

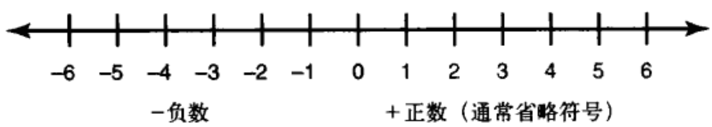
数值是计算机系统最常用的数据类型。与其他数据类型不同的是，不必把数字数据映射到二进制代码。因为二进制也是一种记数系统，所以在数字信息和计算机存储的表示它们的二进制数值之间有种自然对应的关系。通常对正整数来说都是这样的。在第2章关于二进制系统和其他等价记数系统的讨论中，我们介绍了整数转换的问题。但是，还有其他关于数字数据表示法的问题需要考虑。整数不过是数字数据的一部分。这一节将讨论负数和非整数数值的表示法。

3.2.1 负数表示法

负数只是前面带有负号的数吗？也许吧。这当然是看待负数的有效方式之一。让我们来探讨关于负数的问题，讨论在计算机上表示负数的适当方式。

符号数值表示法

从初次在中学学习负数开始，你就使用过数的**符号数值表示法**。在传统的十进制系统中，数值之前带有符号（+或-），只不过正号通常被省略。符号表示了数所属的分类，数字表示了它的量值。标准的实数直线图如下，其中负号表示该数位于0的左侧，正数位于0的右侧。



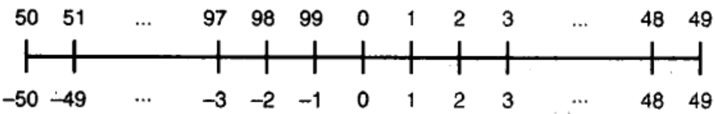
> **符号数值表示法**（signed-magnitude representation）：符号表示数所属的分类（正数或负数），值表示数的量值的数字表示法。

对带符号的整数执行加法和减法操作，可以被描述为向一个方向或另一个方向移动一定的数字单位。求两个数的和，即找到第一个数的刻度，然后向第二个数的符号所示的方向移动指定的数字单位。执行减法的方式一样，即按照符号所示的方向沿着实数直线图移动指定的单位。在中学，即使不使用实数直线图，你都能够很快掌握加法和减法运算。

符号数值表示法有一个问题，即表示0的方法有两种。一种是+0，一种是-0。我们不会对-0感到迷惑，忽略它即可。但是，在计算机中，0的两种表示法却会引起不必要的麻烦，所以还有其他表示负数的方法。让我们来分析另一种负数表示法。

定长量数

如果我们只允许用定量的数值，那么可以用一半数表示正数，另一半数表示负数，符号由数的量值决定。例如，假定能够表示的最大十进制数是99，那么可以用1到49表示正数1到49，用50到99表示负数−50到−1。这种表示法的实数直线图如下所示，它标示了上面的数对应的负数。



在这种模式下执行加法，只需要对两个数求和，舍弃进位即可。求两个正数的和应该没有什么问题，让我们来尝试求一个正数加一个负数、一个负数加一个正数以及两个负数相加。下表分别列出了用符号数值表示法和用这种模式执行的加法运算。（注意，进位被舍弃了。）

符号数值表示法	新 模 式
5 + −6 −1	5 + 94 99
−4 + 6 2	96 + 6 2
−2 + −4 −6	98 + 96 94

用这种模式表示的负数的减法运算又如何呢？关键是加法和减法之间的关系，即A − B = A + (− B)。从一个数中减去另一个数，等价于给第一个数加上第二个数的负数。

符号数值表示法	新 模 式	加 负 数
−5 − 3 −8	95 − 3	95 + 97 92

在这个例子中，我们假定只有100个数值，这个数量非常小，使我们能够用实数直线图来计算一个数的负数表示法。不过，要计算负数表示法，可以采用下列公式。

$$\text{Negative}(I) = 10^k - I\text{，其中}k\text{是数字个数}$$

在两位数字表示法中，求−3的表示法的公式如下：

$$-(3) = 10^2 - 3 = 97$$

在三位数字表示法中，求−3的表示法的公式如下：

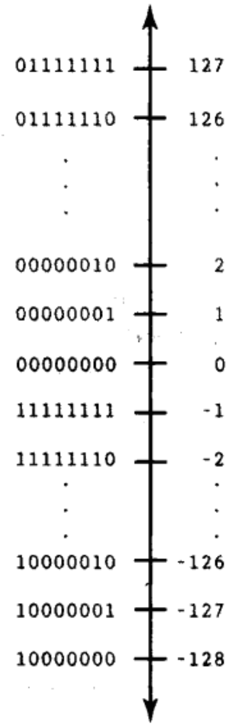
$$-(3) = 10^3 - 3 = 997$$

这种负数表示法称为**十进制补码**。虽然人类以符号和量值表示数字，但在电子计算中，补码在某些方面更方便。由于现代计算机存储任何数据采用的都是二进制，所以我们采用与十进制补码等价的二进制补码。

> **十进制补码**（ten's complement）：一种负数表示法，负数I用10的k次幂减I表示。

二进制补码

假定数字只能用八位表示。为了便于查看长的二进制数，我们把实数直线图绘制成垂直的。



加法和减法的运算方式与十进制补码一样：

− 127	10000001
+ 1	00000001
− 126	10000010

使用这种表示法，负数的最左边一位总是1。因此，在二进制补码中，你可以立刻识别出一个数是正数还是负数。

数字溢出

当我们分配给结果的位数存不下计算出的值时，将发生**溢出**。例如，如果存储每个值使用八位，那么127加3的结果将溢出：

```
  01111111
+ 00000011
  10000010
```

> **溢出**（overflow）：给结果预留的位数存不下计算出的值的状况。

在我们的模式中，10000010表示－126，而不是＋130。但是，如果表示的不是负数，这个结果将是正确的。

溢出，是把无限的世界映射到有限的机器上会发生的典型问题。无论给一个数字分配多少位，总有潜在的表示这些位不能满足的数的需要。如何解决溢出问题，不同的计算机硬件和不同的程序设计语言有自己独特的方法。

3.2.2 实数表示法

在计算中，我们把非整数的值称为实值。我们根据实数在计算机中的用途，把它定义为可能具有小数部分的值。也就是说，实数具有整数部分和小数部分，每个部分都可能是0。例如，104.32、0.999999、357.0和3.14159都是十进制实数。

我们在第2章中介绍过，用数字的位置表示数值，位值是由基数决定的。在十进制中，小数点左侧的位值有1、10、100，依此类推。它们都是基数的幂，从小数点开始向左，每一位升高一次幂。小数点右侧的位值也是这样得到的，只不过幂是负数。所以，小数点右侧的位置是十分位（10^{-1}或十分之一）、百分位（10^{-2}或百分之一），依此类推。

二进制采用的是同样的规则，只是基数为2。由于处理的不是十进制数，所以在英语中，用radix point（**小数点**）代替十进制中的radix point（小数点），任何记数系统都可以使用这个术语。在二进制中，小数点右侧的位置是二分位（2^{-1}或二分之一）、四分位（2^{-2}或四分之一），依此类推。

那么如何在计算机中表示一个实值呢？我们把实数存储为一个整数加指示小数点位置的信息。也就是说，任何实值都可以由三个属性描述，即符号（正号或负号）、尾数和指数，尾数由该数值中的数字构成，假定小数点在其右边，指数确定了小数点相对于尾数的位移。十进制的实数可以用下列公式定义：

$$符号 \times 尾数 \times 10^{exp}$$

这种表示法称为**浮点表示法**，因为数字的个数是固定的，但是小数点却是浮动的。在用浮点形式表示的数值中，正指数将把小数点向右移，负指数将把小数点向左移。

> **小数点**（radix point）：在记数系统中，把一个实数分割成整数部分和小数部分的点。
> **浮点表示法**（floating point）：标明了符号、尾数和指数的实数表示法。

让我们来看看如何把实数常用的十进制表示法转换成浮点表示法。例如，考虑实数148.69，符号是正号，小数点右边有两位数字，因此，指数是 -2，浮点表示法即 $14\ 869 \times 10^{-2}$。表3-1给出了其他例子。为了便于讨论，我们假设只能表示五位数字。

表3-1　十进制表示法和浮点表示法表示的（五位数字）值

实　值	浮 点 值	实　值	浮 点 值
12001.00	$12001*10^{0}$	-123.10	$-12310*10^{-2}$
-120.01	$-12001*10^{-2}$	155555000.00	$15555*10^{3}$
0.12000	$12000*10^{-5}$		

如何把浮点数转换回十进制表示法呢？基数上面的指数说明了小数点要移动多少位。如果指数是负数，小数点要向左移；如果指数是正数，小数点要向右移。对表3-1中的浮点数应用这个规则。

注意表3-1中的最后一个例子，它丢失了信息。因为我们只保存五位数字来表示有效数字（尾数），所以这个值的整数部分在浮点表示法中没有被精确地表示出来。

同样地，下面的公式定义了一个二进制浮点值：

$$符号 \times 尾数 \times 2^{exp}$$

注意，只有基数改变了。当然，尾数只能包含二进制数字。要在计算机上存储二进制的浮点数，可以保存定义它的三个值。例如，根据一条通用准则，如果用64位存储一个浮点值，那么其中1位存储符号，11位存储指数，52位存储尾数。当一个值用于计算或显示时，都会采用这种格式。

如果一个数不完整，那么如何才能得到尾数的正确值呢？在第2章中，我们讨论过如何把自然数从一种记数系统转换到另一种记数系统。这里，我们用十进制的例子说明了在计算机中如何表示实数。我们知道，计算机中的所有数值都是用二进制表示的。那么如何把十进制数的小数部分转换成二进制的呢？

把一个整数从十进制转换成其他数制，需要用新基数除这个数，余数是结果左边的下一位数字，商是新的被除数，整个过程直到商为0终止。转换小数部分的操作是类似的，只不过不是用新基数除这个数，而是用新基数乘它。乘法的进位将成为答案左边的下一位数字，乘法结果中的小数部分将成为新的被乘数，整个过程直到乘法结果中的小数部分为0截止。让我

们把0.75转换成二进制的。

$$0.75 * 2 = 1.50$$
$$0.50 * 2 = 1.00$$

因此，十进制中的0.75是二进制中的0.11。让我们再做一个转换。

$$0.435 * 2 = 0.870$$
$$0.870 * 2 = 1.740$$
$$0.740 * 2 = 1.480$$
$$0.480 * 2 = 0.960$$
$$0.960 * 2 = 1.920$$
$$0.920 * 2 = 1.840$$
$$\cdots$$

因此，十进制中的0.435是二进制中的011011…。小数部分会变成0吗？继续乘下去，看看结果如何。

下面，让我们看一个完整的转换过程，把十进制的20.25转换成二进制的。首先，转换20。

$$\begin{array}{r} 10 \\ 2\overline{)20} \\ \underline{20} \\ 0 \end{array} \qquad \begin{array}{r} 5 \\ 2\overline{)10} \\ \underline{10} \\ 0 \end{array} \qquad \begin{array}{r} 2 \\ 2\overline{)5} \\ \underline{4} \\ 1 \end{array} \qquad \begin{array}{r} 1 \\ 2\overline{)2} \\ \underline{2} \\ 0 \end{array} \qquad \begin{array}{r} 0 \\ 2\overline{)1} \\ \underline{0} \\ 1 \end{array}$$

20在二进制中等价于10100。现在我们来转换小数部分：

$$0.25 * 2 = 0.50$$
$$0.50 * 2 = 1.00$$

因此，十进制的20.25在二进制中是10100.01。

科学记数法可能是你已经熟悉的术语，所以我们在这里只简要介绍一下。科学记数法是浮点表示法的一种形式，其中，小数点总在最左边数字的右侧。也就是说，整数部分只有一位。在许多程序设计语言中，如果在输出一个大的实数值时没有指定输出格式，那么这个值将以科学记数法输出。因为早期的机器不能输出指数，所以用字母“E”代替。例如，在科学记数法中，12001.32708将被写为1.200132708E+4。

> **科学记数法**（scientific notation）：另一种浮点表示法。

3.3　文本表示法

一个文本文档可以被分解为段落、句子、词和最终的字符。用数字形式表示文本文档，只需要表示每个可能出现的字符。文档是连续（模拟）的实体，独立的字符则是离散的元素，它们才是我们要表示并存储在计算机内存中的。

现在，我们应该区分文本表示法的基本的想法和文字处理的概念。当我们用Microsoft Word这样的字处理程序创建文档时，可以设置各种格式，包括字体、页边距、制表位、颜色，等等。许多字处理程序还允许在文档中加入自选图形、公式和其他元素。这些额外信息与文本存储在一起，以便文档能够正确地显示和打印出来。但核心问题是如何表示字符本身，因

此，目前我们把重点放在表示字符的方法上。

要表示的字符数是有限的。一种表示字符的普通方法是列出所有字符，赋予每个字符一个二进制字符串。要存储一个特定的字母，我们将保存它对应的位串。

那么我们需要表示哪些字符呢？在英语中，有26个字母，但必须有区别地处理大写字母和小写字母，所以实际上有52个字母。与数字（0、1到9）一样，各种标点符号也需要表示。即使是空格，也需要有自己的表示法。那么对于非英语的语言又如何呢？一旦你考虑到这一点，那么我们想表示的字符数就会迅速增长。记住，我们在本章的前面讨论过，要表示的状态数决定了需要多少位来表示一种状态。

字符集只是字符和表示它们的代码的清单。这些年来，出现了多种字符集，但只有少数的几种处于主导地位。在计算机制造商就关于使用哪种字符集达成一致后，文本数据的处理变得容易多了。在接下来的小节中，我们将介绍两种字符集，即ASCII字符集和Unicode字符集。

> **字符集**（character set）：字符和表示它们的代码的清单。

3.3.1 ASCII字符集

ASCII是美国信息互换标准代码（American Standard Code for Information Interchange）的缩写。最初，ASCII字符集用7位表示每个字符，可以表示128个不同的字符。每个字节中的第八位最初被用作校验位，协助确保数据传输正确。之后，ASCII字符集进化了，用8位表示每个字符。这个8位版本的正式名字是Latin-1扩展ASCII字符集。该扩展字符集可以表示256个字符，包括一些重点字符和几个补充的特殊符号。图3-5展示了完整的ASCII字符集。

右边的数字 / 左边的数字	ASCII 0	1	2	3	4	5	6	7	8	9
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT
1	LF	VT	FF	CR	SO	SI	DLE	DC1	DC2	DC3
2	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS
3	RS	US	□	!	"	#	$	%	&	'
4	(	)	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	\	]	^	_	`	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{	|	}	~	DEL		

图3-5 ASCII字符集

这个图表中的代码是用十进制表示的，但计算机存储这些代码时，将把它们转换成相应的二进制数。注意，每个ASCII字符都有自己的顺序，这是由存储它们所用的代码决定的。每个字符都有一个相对于其他字符的位置（在其他字符之前或之后）。这个属性在许多方面都很有用。例如，可以利用字符代码对一组单词按照字母顺序排序。

这个图表中的前32个ASCII字符没有简单的字符表示法，不能输出到屏幕上。这些字符是为特殊用途保留的，如回车符和制表符，处理数据的程序会用特定的方式解释它们。

3.3.2 Unicode字符集

ASCII字符集的扩展版本提供了256个字符，虽然足够表示英语，但是却无法满足国际需要。这种局限性导致了Unicode字符集的出现，这种字符集具有更强大的国际影响。

Unicode的创建者的目标是表示世界上使用的所有语言中的所有字符，包括亚洲的表意符号。此外，它还表示了许多补充的专用字符，如科学符号。

为了实现这个目标，Unicode字符集使用16位表示每个字符。因此，Unicode字符集能够表示2^{16}个字符，即6万5千多个字符。这个数字远远大于扩展ASCII字符集表示的字符个数256。

现在，Unicode字符集大受欢迎，许多程序设计语言和计算机系统都采用它。但是，Unicode字符集本身仍在发展中。并非所有可用的代码都被赋予了特定的字符。图3-6展示了Unicode字符集当前具有的几个字符。

为了保持一致，Unicode字符集被设计为ASCII的超集。也就是说，Unicode字符集中的前256个字符与扩展ASCII字符集中的完全一样，表示这些字符的代码也一样。因此，即使底层系统采用的是Unicode字符集，采用ASCII值的程序也不会受到影响。

代　码	字　符	来　源
0041	A	English (Latin)
042F	Я	Russian (Cyrillic)
0E09	ฉ	Thai
13EA	Ꮺ	Cherokee
211E	℞	Letterlike Symbols
21CC	⇌	Arrows
282F	⠯	Braille
345F	㑟	Chinese/Japanese/Korean (Common)

图3-6　Unicode字符集中的几个字符

3.3.3 文本压缩

字母信息（文本）是一种基本数据类型。因此，找到存储这种信息以及有效地在两台计算机之间传递它们的方法是很重要的。下面的小节将分析三种文本压缩类型：

- 关键字编码
- 行程长度编码
- 赫夫曼编码

在本章后面的小节中我们还会谈到，这些文本压缩方法的基本思想也适用于其他类型的数据。

关键字编码

想想你在英语中使用“the”、“and”、“which”、“that”和“what”的频率。如果这些单词占用更少的空间（即用更少的字符表示），文档就会减小。即使每个单词节省的空间都很少，但它们在典型的文档中太常用，所以节省出的总空间还是很可观的。

一种相当直接的文本压缩方法是**关键字编码**，它用单个字符代替了常用的单词。要解压这种文档，需要采用压缩的逆过程，即用相应的完整单词替换单个字符。

> **关键字编码**（keyword encoding）：用单个字符代替常用的单词。

例如，假设我们用下列图表对几个单词编码：

单 词	符 号	单 词	符 号
as	^	must	&
the	~	well	%
and	+	these	#
that	$		

让我们对下列段落编码：

The human body is composed of many independent systems, such as the circulatory system, the respiratory system, and the reproductive system. Not only must all systems work independently, they must interact and cooperate as well. Overall health is a function of the well-being of separate systems, as well as how these separate systems work in concert.

编码后的段落如下：

The human body is composed of many independent systems, such ^ ~ circulatory system, ~ respiratory system, + ~ reproductive system. Not only & each system work independently, they & interact + cooperate ^ %. Overall health is a function of ~ %-being of separate systems, ^ % ^ how # separate systems work in concert.

原始段落总共有352个字符，包括空格和标点。编码后的段落包括317个字符，节省了35个字符。这个例子的压缩率是317/352，或约为0.9。

关键字编码有几点局限性。首先，用来对关键字编码的字符不能出现在原始文本中。例如，如果原始段落中包括“$”，那么生成的编码就会有歧义。我们不知道“$”表示的是单词“that”还是真正的美元符号。这限制了能够编码的单词数和要编码的文本的特性。

呼叫所有汽车

许多城市常见的交通监控中心现在在尝试预测何时何地会出现交通堵塞。Microsoft的员工正在测试一个叫做JamBayes的交通预测系统，它能覆盖整个Redmond。用户只需要报告他们想行驶的路线，就会收到关于即将出现的交通堵塞的警告消息。JamBayes采用了名为贝叶斯模型的数学方法，能够把实时的交通数据与历史趋势、天气信息和日期事件表（如假期）联系在一起。该系统的准确率可达到75%。

此外，示例中的单词“The”没有被编码为字符“~”，因为“The”与“the”不是同一个单词。记住，在计算机上存储的字母，它的大写版本和小写版本是不同的字符。如果想对“The”编码，必须使用另一个符号，或者采用更加复杂的替换模式。

最后，不要对“a”和“I”这样的单词编码，因为那不过是用一个字符替换另一个字符。单词越长，每个单词的压缩率就越高。遗憾的是，常用的单词通常都比较短。另一方面，有些文档使用某些单词比使用其他单词频繁，这是由文档的主题决定的。例如，在我们的示例中，如果对单词“system”编码，将节省很多空间，但在通常情况下，并不值得对它编码。

关键字编码的一种扩展是用特殊字符替换文本中的特定模式。被编码的模式通常不是完整的单词，而是单词的一部分，如通用的前缀和后缀“ex”、“ing”和“tion”。这种方法的一个优点是被编码的模式通常比整个单词出现的频率更高。但缺点同前，即被编码的通常是比较短的模式，对每个单词来说，替换它们节省的空间比较少。

行程长度编码

在某些情况下，一个字符可能在一个长序列中反复出现。在英语文本中，这种重复不常见，但在大的数据流（如DNA序列）中，这种情况则经常出现。一种名为**行程长度编码**的文本压缩方法利用了这种情况。行程长度编码有时又称为*迭代编码*。

在行程长度编码中，重复字符的序列将被替换为*标志字符*，后面加重复字符和说明字符重复次数的数字。例如，下面的字符串由7个A构成：

AAAAAAA

如果用*作为标志字符，这个字符串可以被编码为：

*A7

> **行程长度编码**（run-length encoding）：把一系列重复字符替换为它们重复出现的次数。

标志字符说明这三个字符的序列应该被解码为相应的重复字符串，其他文本则按照常规处理。因此，下列编码字符串：

*n5*x9ccc*h6 some other text *k8eee

将被解码为如下的原始文本：

nnnnnxxxxxxxxxccchhhhhh some other text kkkkkkkkeee

原始文本包括51个字符，编码串包括35个字符，所以这个示例的压缩率为35/51或约为0.68。

注意，这个例子中有三个重复的c和三个重复的e都没有编码。因为需要用三个字符对这样的重复序列编码，所以对长度为2或3的字符串编码是不值得的。事实上，如果对长度为2的重复字符串编码，反而会使结果串更长。

因为我们用一个字符记录重复的次数，所以看来不能对重复次数大于9的序列编码。但是，在某些字符集中，一个字符是由多个位表示的。例如，字符“5”在ASCII字符集中表示为53，这是一个八位的二进制字符串00110101。因此，我们将次数字符解释为一个二进制数，而不是解释为一个ASCII数字。这样一来，能够编码的重复字符的重复次数可以是0到255间的任何数，甚至可以是4到259之间的任何数，因为长度为2或3的序列不会被编码。

赫夫曼编码

另一种文本压缩方法是**赫夫曼编码**，以它的创建者David Huffman博士的名字命名。文本中很少使用字母“X”，那么为什么要让它占用的位数与其他常用字符一样呢？赫夫曼编码使用不同长度的位串表示每个字符，从而解决了这个问题。也就是说，一些字符由5位编码表示，一些字符由6位编码表示，还有一些字符由7位编码表示，等等。这种方法与字符集的概念相反，在字符集中，每个字符都由定长（如8位或16位）的位串表示。

这种方法的基本思想，是用较少的位表示经常出现的字符，而将较长的位串留给不经常出现的字符，这样表示的文档的整体大小将比较小。

> **赫夫曼编码**（Huffman encoding）：用变长的二进制串表示字符，使常用的字符具有较短的编码。

例如，假设用下列赫夫曼编码来表示一些字符：

赫夫曼编码	字　符	赫夫曼编码	字　符
00	A	111	R
01	E	1010	B
100	L	1011	D
110	O		

那么单词DOORBELL的二进制编码如下：

1011110110111101001100100

如果使用定长位串（如8位）表示每个字符，那么原始字符串的二进制形式应该是8个字符×8位=64位。而这个字符串的赫夫曼编码的长度是25位，压缩率为25/64，或约为0.39。

那么解码过程是怎样的呢？在使用字符集时，只要把二进制串分割成8位或16位的片段，然后查看每个片段表示的字符即可。在赫夫曼编码中，由于编码是变长的，我们不知道每个字符对应多少位编码，所以看似很难将一个字符串解码。其实，创建编码的方式已经消除了这种潜在的困惑。

赫夫曼编码的一个重要特征是用于表示一个字符的位串不会是表示另一个字符的位串的前缀。因此，在从左到右扫描一个位串时，每当发现一个位串对应于一个字符，那么这个位串就一定表示这个字符，该位串不可能是更长位串的前缀。

例如，如果下列位串是用上面的表创建的：

1010110001111011

那么它只会被解码为单词BOARD，没有其他的可能性。

那么，赫夫曼编码是如何创建的呢？虽然创建赫夫曼编码的详细过程不属于本书的介绍范围，但是我们可以讨论一下要点。由于赫夫曼编码用最短的位串表示最常用的字符，所以首先需要列出要编码的字符的出现频率。出现频率可以是字符在某个特定文档中出现的次数（如352个E、248个S等），也可以是字符在来自特定领域的示例文本中出现的次数。频率表则列出了字母在一种特定语言（如英语）中出现的频率。使用这些频率，可以构建一种二进制代码的结构。创建这种结构的方法确保了最常用的字符对应最短的位串。

3.4 音频信息表示法

当一系列空气压缩震动我们的耳膜时，给我们的大脑发送了一个信号，我们就感觉到了声音。因此，声音实际上是由与我们的耳膜交互的声波定义的。请参阅图3-7。要表示声音，必须正确地表示声波。



图3-7　震动我们耳膜的声波

一个立体声系统通过把电信号发送到一个扬声器来制造声音。这种信号是声波的模拟表示法。信号中的电压按声波的正比例变化。扬声器接收到信号后，将引起膜震动，依次引起空气震动（创建了声波），从而引起耳膜震动。创建的声波有可能与扬声器初始接收到的完全一样，或者至少能让听众满意。

要在计算机上表示音频信息，必须数字化声波，把它分割成离散的、便于管理的片段。方法之一是真正数字化声音的模拟表示法。也就是说，采集表示声波的电信号，用一系列离散的数值表示它。

纷乱的字符集

1960年，《Communications of the ACM》上的一篇文章对使用中的字符集做了一份调查，介绍了60种不同的字符集。在IBM系列的计算机中，存在9种内容和顺序都不一样的字符集。[1]

模拟信号是随电压连续变化的。要数字化这种信号，需要周期性地测量信号的电压，记录合适的数值，这一过程称为*采样*，最后得到的不是连续的信号，而是表示不同电压电平的一系列数字。

用存储的电压值创建一个新的连续电信号，可以使声音再生。这里有一个假设，即原始信号中的电压电平是均匀地从一个存储的电压值变化到下一个电压值的。如果在短时期内采到了足够的样本，这种假设是合理的。但毫无疑问，采样过程会丢失信息，如图3-8所示。

一般说来，采样率在每秒40 000次左右就足够创建合理的声音复制品。如果采样率低于这个值，人耳听到的声音会失真。较高的采样率生成的声音质量较好，但到达某种程度后，额外的数据都是无关的，因为人耳分辨不出其中的差别。声音的整体效果是受很多因素影响的，包括设备的质量、声音的类型和人的听力等。

塑胶唱片是声波的模拟表示法。电唱机（唱机转盘）的唱针在唱片的螺旋形凹槽中上下伸缩。唱针的上下伸缩模拟了表示声音的信号的电压变化。

另一方面，激光唱盘（CD）则存储了数字化的音频信息。CD的表面是用显微镜可见的凹点，表示二进制数字。低强度的激光将指向唱盘。如果唱盘表面是光滑的，激光的反射强烈，如果唱盘表面有凹痕，激光的反射就比较少。接收器将分析反射的强度，生成适当的二进制数字串，这是信号被数字化后存储的数字电压值。该信号将被重现，发送给扬声器。图3-9展示了这一过程。

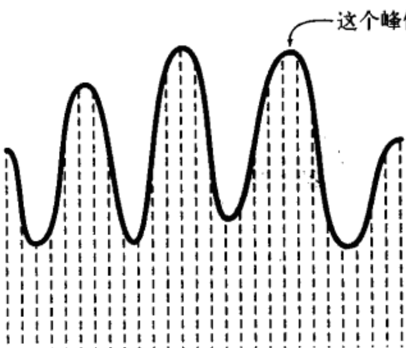


图3-8 音频信号的采样

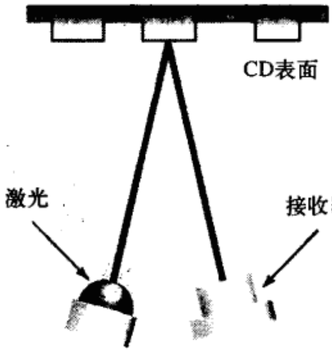


图3-9 读取二进制信息的CD播放器

3.4.1 音频格式

在过去几年中，出现了多种流行的音频信息格式，包括WAV、AU、AIFF、VQF和MP3等。尽管所有格式都是基于从模拟信号采样得到的电压值的，但是它们格式化信息细节的方式不同，采用的压缩方法也不同。

当前，处于统治地位的压缩音频数据格式是MP3。MP3的盛行，主要源于它的压缩率比同时期的其他格式的压缩率高。即使将来能证明其他格式更有效，但现在MP3是大众的最爱。在1999年中期，“MP3”这个词的检索频率远远高于其他词，而且现在还在盛行。让我们看看MP3格式的细节。

3.4.2 MP3音频格式

MP3是MPEG-2 audio layer 3的缩写。MPEG是Moving Picture Experts Group（运动图像专家组）的缩写，这是为数字音频和视频开发压缩标准的国际委员会。

MP3格式使用有损压缩和无损压缩两种压缩方法。首先，它将分析频率展开，与人类心理声学的数学模型进行比较，然后舍弃那些人类听不到的信息，再用赫夫曼编码进一步压缩得到的位流。

网络上有很多可用的软件工具能帮助你创建MP3文件。这些工具通常要求在把数据转换成MP3格式之前，录制品是用某种通用格式（如WAV）存储的，这样可以使文件大大减小。

解释和播放MP3文件的播放器有很多。MP3播放器既可以是纯粹的计算机软件，也可以像流行的iPod一样，是一种专用的硬件设备，能够存储和播放MP3文件。其实，现在很多CD播放器也可以播放MP3格式的文件。大多数MP3播放器允许用户用各种方式组织他们的文件，能够显示特定文件的各种信息以及它们对应的图形。

3.5 图像和图形的表示法

在讨论图像（如照片）和图形（如线条画）的表示法及压缩方法时，它们有些共同点。首先，我们来看看表示颜色的基本方法，然后再介绍各种数字化和表示视频信息的方法。

3.5.1 颜色表示法

颜色是我们对到达视网膜的各种频率的光的感觉。我们的视网膜有三种颜色感光视锥细胞，负责接收不同频率的光。这些感光器分类分别对应于红、绿和蓝三种颜色。人眼可以觉察的其他颜色都能由这三种颜色混合而成。

在计算机中，颜色通常用RGB（red-green-blue）值表示，这其实是三个数字，说明了每种原色的相对份额。如果用0到255之间的数字表示一种元素的份额，那么0表示这种颜色没有参与，255表示它完全参与其中。例如，RGB值（255，255，0）最大化了红色和绿色的份额，最小化了蓝色的份额，结果生成的是嫩黄色。

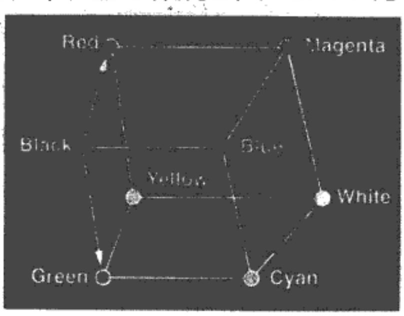


图3-10 三维色空间

RGB值的概念导致了三维“色空间”。图3-10展示了一种显示色空间的方法。

用于表示颜色的数据量称为色深度。通常用表示颜色的位数来表示色深度。增强彩色指

色深度为16位的颜色，RGB值中的每个数字由5位表示，剩下的一位有时用于表示透明度。**真彩色**指色深度为24位的颜色，RGB值中的每个数字由8位表示，即每个数所属的范围是0～255，这样能够生成1670万种以上的颜色。

下表展示了一些真彩色的RGB值和它们表示的颜色：

RGB值			颜 色
红 色	绿 色	蓝 色	
0	0	0	黑色
255	255	255	白色
255	255	0	黄色
255	130	255	粉色
146	81	0	棕色
157	95	82	紫色
140	0	0	栗色

24位真彩色提供的颜色比人眼能够分辨的颜色多。此外，显示器能显示的颜色也受限于特定的色深度。为了使显示器显示的颜色减少到256色，程序指定的任何颜色都会被映射到硬件能够显示的调色板中与之最接近的一种颜色。图3-11显示了这种受限制的调色板。当想要显示的颜色与硬件能够显示的颜色之间差别太大时，显示的结果通常都不令人满意。令人欣慰的是，大多数现代的显示器都提供了足够的颜色范围，大大减少了这种问题。

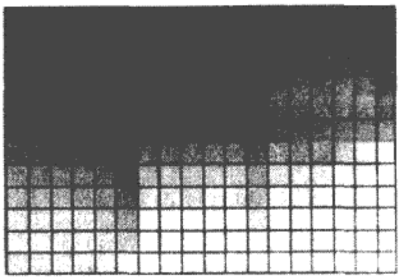


图3-11 受限的调色板

3.5.2 数字化图像和图形

照片是图像的模拟表示。它的表面是连续的，一种颜色的色度与另一种颜色的混合在一起。数字化一幅图像是把它表示为一套独立的点，这些点称为**像素**，代表图像的元素。每个像素由一种颜色构成。表示一幅图像使用的像素个数称为**分辨率**。如果使用了足够的像素（高分辨率），把它们按正确的顺序并排排列，就可以瞒过人眼，使人们认为看到的是连续的图像。图3-12展示了一个数字化的图像，它的一小部分被放大，显示出了独立的像素。

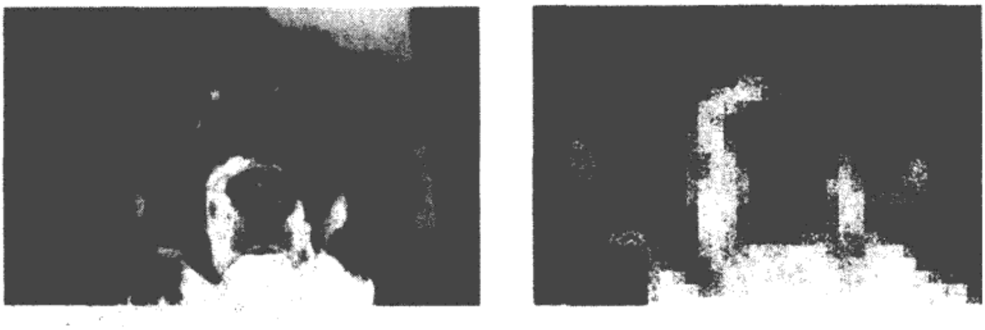


图3-12 由许多独立像素构成的数字化图像

> **像素**（pixels）：用于表示图像的独立点，代表图像的元素。

逐个像素存储图像信息的方法称为**光栅图形格式**。目前流行的几种光栅文件格式有位图（BMP）、GIF和JPEG。

> **分辨率**（resolution）：用于表示图像的像素个数。
> **光栅图形格式**（raster-graphics format）：逐个像素存储图像信息的格式。

位图文件是最简单的图形表示之一。除了一些管理细节外，位图文件只包括图像的像素颜色值，按照从左到右、从上到下的顺序存放。虽然位图文件支持24位的真彩色，但是通常会指定色深度，以减小文件。可以使用本章前面介绍过的行程长度编码来压缩位图文件。

CompuServe于1987年开发的GIF（Graphics Interchange Format，图形交换格式）把图像中可用的颜色数量限制在256种颜色。也就是说，GIF图像只能由256种颜色构成，不过不同的GIF图像可以由包含256种颜色的不同颜色集构成。这种技术叫做*索引颜色*，由于要引用的颜色少，所以生成的结果文件就比较小。如果需要使用更少的颜色，那么可以采用需要更少位数的色深度。GIF文件最适合用于颜色较少的图形和图像，因此，它是存放线条图像的首选格式。有一种GIF格式版本还可以通过存储一系列能在浏览器这样的程序中连续显示的图像来定义动画。

JPEG格式利用了人眼的特性。人眼对亮度和颜色的渐变比对它们的迅速改变敏感。因此，JPEG格式保存了短距离内色调的平均值。JPEG格式被看作存储照片颜色图像的首选格式。它采用的压缩模式相当复杂，有效地减小了生成的文件。

PNG是Portable Network Graphics（可移植的网络图像文件格式）的缩写。PNG格式的设计者是想用PNG格式改进当前GIF格式，从而最终取代它。PNG图像的压缩效果通常比GIF图像更好，同时提供的色深度范围也更广。不过，PNG格式不支持动画，也不像GIF格式那样广受支持。

Bob Bemer



从1945年起，Bob Bemer就常出现在计算圈中。他的供职履历读起来就像20世纪后半期有影响力的计算技术公司的清单。他曾经在Douglas Aircraft、RKO Radio Pictures、the Rand Corporation、Lockheed Aircraft、Marquardt Aircraft、Lockheed Missiles and Space、IBM、Univac Division of Sperry Rand、Bull General Electric（Paris）、GTE、Honeywell工作过，最后成立了自己的软件公司Bob Bemer Software。

在Bemer的简历上出现过很多飞机制造厂商，这一点都不奇怪，因为他的专业是数学，而且从Curtiss-Wright技术学院（1941）获得了航空工程学的毕业证书。在计算历史的早期，飞机制造商是工业界使用计算机的先驱。

在他的职业生涯中，Bemer致力于程序设计语言的开发。他开发了早期的FORTRAN编译器FORTRANSIT，还积极参与COBOL语言和CODASYL语言（一种早期的数据库建模和管理方法）的开发。此外，他还负责SIMULA语言开发基金的授权工作，这是一种引入了多个面向对象特性的模拟语言。

Bemer还是为新兴的计算业制定全球标准的委员会的成员。他是IFIP Computer Vocabulary

Committee的美国代表，ISO/TC97/SC5 on Common Programming Languages的主席，X3/SPARC Study Group on Text Processing的主席。

但是，Bemer最著名的工作是关于ASCII编码这种8位PC的标准内部码。当初，Bemer认识到如果一台计算机要与另一台计算机通信，它们需要传送文本信息的标准代码。Bemer发布了关于60多种计算机代码的调查报告，从而说明了对标准代码的需要。他拟订了标准委员会的工作计划，促使美国标准代码与国际代码相对应，编写了大量关于编码的文献，为ASCII的备用符号和适用其他语言的控制集争取到了正式的注册。

也许Bemer最重要的贡献是提出了转义符这种概念。转义符将通知系统，转义符后的字符不再使用它的标准含义。例如，ESC(N通知系统下列字符是与ASCII字符等价的Cyrillic字符。

1991年10月发布了16位编码的第一个版本Unicode。促使这种扩大的编码出现的原因有两个，一是16位计算机体系结构越来越流行了，二是Internet和万维网的盛行，需要一种能直接包含全世界各种字母的编码。但是，ASCII并没有退出历史舞台，它是Unicode的一个子集。

2003年5月，Bemer得到了IEEE Computer Society颁发的计算机先驱奖（Computer Pioneer Award），以表彰他"通过ASCII、ASCII备用字符集和转义序列为满足世界对各种字符集和符号的需要"所做出的贡献。

Bob Bemer于2004年6月22日在位于得克萨斯州Possum Kingdom Lake的家中逝世。

http://www.bobbemer.com/AWARD.HTM

3.5.3 图形的矢量表示法

矢量图形是另一种表示图像的方法。矢量图形格式不像光栅图形那样把颜色赋予像素，而是用线段或几何形状描述图像。矢量图形是一系列描述线段的方向、线宽和颜色的命令。由于不必记录所有的像素，所以采用这种格式的文件一般比较小。图像的复杂度（如图像中的项目个数）决定了文件的大小。

> **矢量图形**（vector graphics）：用线段或几何形表示图像的方法。

光栅图形（如GIF图形）要获得不同的大小和比例必须进行多次编码。矢量图形则可以通过数学计算调整大小，这些改变可以根据需要动态地计算。

但是，矢量图形不适用于表示真实世界的图像。JPEG图像是表示真实世界图像的首选，矢量图形则适用于艺术线条和卡通绘画。

当前，网络上最流行的矢量格式是Flash。Flash图像存储为二进制格式，创建Flash图像需要专用的编辑器。一种新的矢量格式SVG（Scalable Vector Graphics，可缩放矢量图形）正在开发中，它是用纯文本表示的。一旦SVG格式完成了，矢量图形可能会成为网络成像的流行方法。

爱因斯坦对电报的描述

"你看，有线电报就像一只体型非常非常长的猫，"阿尔伯特·爱因斯坦（Albert Einstein）这样解释道，"你在纽约拉住了它的尾巴，却听到它的脑袋在洛杉矶叫……无线电收发报机也是这样，你在这里发出了信号，他们可以在那里收到。唯一的区别是没有猫。"你认为他会怎样描述计算机呢？

3.6 视频表示法

视频信息的捕捉和压缩方法使它成了最复杂的信息类型之一。视频片段包含许多压缩的静态图像。网络上充满了结构拙劣、难以理解的视频片段。随着视频压缩技术（即视频编译码器）的发展，这种情况在未来的几年中可能会得到改善。

视频编译码器

编译码器（codec）表示压缩器/解压缩器（COmpressor/DECompressor）。**视频编译码器**指用于缩减电影大小的方法，使它能够在计算机或网络上播放。几乎所有的视频编译码器都采用有损压缩，最小化与视频相关的数据量，因此，压缩的目标不是舍弃影响观众视觉的信息。

> **视频编译码器**（video codec）：用于缩减电影大小的方法。

大多数编译码器是面向块的，也就是说，视频的每个帧将被分成一组矩形块。各个编译码器的不同之处在于如何对这些块编码。有些视频编译码器完全是由软件实现的，而有的则需要专用的硬件。

视频编译码器采用的压缩方式有两种，即时间压缩和空间压缩。**时间压缩**将查找连续帧之间的差别。如果两个帧中的图像大部分都没有改变，那么何必浪费空间来复制所有近似的信息呢？*关键帧*是比较帧之间差别的参照物，它的完整图像都会被保存。对于连续的图像，只保存改变的部分（*增量帧*）。对于帧与帧之间变化不大的视频片段（如几乎没有活动实体的场景），时间压缩是一种有效的方法。

空间压缩将删除一个帧中的冗余信息。空间压缩的基本问题与压缩静态图像时遇到的问题一样。空间视频压缩常把颜色相同的像素（如湛蓝的天空）聚集在块（矩形区域）中，存储的不是每个像素的信息，而是块的颜色和坐标。这种思想与前面介绍的行程长度编码的思想近似。

> **时间压缩**（temporal compression）：根据连续帧之间的差别压缩电影的技术。
> **空间压缩**（spatial compression）：基于静态图像的压缩方法的电影压缩技术。

当今流行的视频编译码器有Sorenson、Cinepak、MPEG和Real Video。关于这些编译码器如何表示和压缩视频，不属于本书介绍的范围。

小结

计算机是多媒体设备，操作的数据从数字到图形，再到视频，无所不包。由于计算机只能操作二进制数值，所以所有类型的数据都必须表示为二进制形式的。数据可以分为两类，连续的（模拟的）和离散的（数字的）。

整数值由它们对应的二进制值表示，负数的表示方法有符号数值表示法和补码表示法。实数由三部分构成，即符号、尾数和指定小数点位置的指数。

字符集是字母与数字字符以及表示它们的代码的清单。最常用的字符集是Unicode（每个字符由16位表示），ASCII是它的子集。使用8位字符集ASCII足够表示英语，但却不足以表示其他语言。压缩文本的方法有很多，可以减小存储文本的空间，减少在机器之间传递文本的

时间。

音频信息被表示为数字化的声波。颜色由三个值表示，每个值说明了红色、蓝色或绿色的份额。表示图像的基本方法有两种，即位图和矢量图形。视频被分割成了一系列静态图像。

道德问题：MGM Studios公司和Grokster有限公司

自从Napster网站出现以来，通过Internet文件共享服务交换私有材料所涉及的版权战争就一直在升级。1999年12月美国唱片工业协会（Recording Industry Association of America，RIAA）对Napster提出控诉，此案件的最终结果导致了Napster网站的转型（Napster从此转为与RIAA合作的“音乐商店”）。其他的文件共享服务提供商，如Morpheus和KaZaA，也支持通过Internet交换包含具有版权的音乐的MP3文件，但是它们从最初就设法避免了Napster的命运。Napster采用的是由服务器、索引和姓名注册表构成的集中式分发点，而Morpheus和KaZaA采用的则是非集中式的文件共享系统，即一种真正的对等网络（P2P）技术。这种结构使得RIAA与面对Napster时的情形不同，它很难挑出法律问题。

在研究了Morpheus、Grokster、Gnutella、KaZaA等其他文件共享站点采用的P2P系统后，RIAA决定采用另一种战略来捕捉使用这种服务在线交换具有版权的音乐的用户。RIAA不再针对P2P服务提供商，而是通过法院向Comcast和Verizon这样的Internet服务提供商（ISP）发传票，以查明它怀疑的下载或交换了具有版权的音乐的用户的真实姓名。大多数ISP按照RIAA的要求提供了用户的姓名，但是Verizon则决定与RIAA对峙法庭，它指出公布名单侵犯了用户的隐私权（参阅Verizon对RIAA的案例）。

当然，关于是否允许通过P2P系统共享有版权的文件的争论并不仅局限于交换有版权的音乐。在RIAA用法律维权的过程中，电影业一直在仔细观察，P2P文件共享系统可以让用户轻松地免费交换电影，这点也引起了电影业的关注。2003年，Metro-Goldwyn-Mayer（MGM）Stuties和其他27家音乐及电影工作室联合对Grokster（KaZaA所属的公司）提出了诉讼。MGM指控这些P2P文件共享服务提供商“协助侵犯版权”，指出它们应对在自己系统上交换的具有版权的资料负法律责任。

Grosker则辩论说，根据美国Sony公司和Universal City Studios公司（1984）的先例，自己的系统是合法的，这个案例决定了不能仅因为可能侵犯版权而禁止一种技术。在这个案例中，Universal提出禁止出售VCR技术，它声称Sony（生产了Betamax家用录像机，成为VHS录像设备的竞争对手）要对直接或间接的侵犯版权行为负责，因为新的技术可能被用于制造电影的非法拷贝。但是高级法院则赞同Sony的观点，即并不能仅因为VCR能够造成真正的版权侵犯就说它违反了法律（这段解释成为著名的“Sony安全港”的先例）。后来法院拒绝了禁止或限制技术发展。由于P2P网络被看作“技术发展”，所以这种技术也应按照Sony的先例处理。但是，MGM却辩论道，通过Grokser的P2P系统交换的资料有90%都是有版权的，这就构成了“真正的”版权侵犯。

最初，一家地区法院不赞同MGM的观点，裁决Grokster不必为分发有版权的资料负法律责任。尽管MGM提出了重审请求，但高一级的Ninth Circuit法院维持了低级法院的裁决。然后，MGM决定向美国最高法院上诉，最高法院同意重审该案。在最高法院辩论时，法官们根据两个显然冲突的原则分为两派，既需要保护像P2P网络这样的新技术，又需要对版权侵犯提供赔偿。

法官不同意有足够的法律先例能保护Grokster逃避版权侵犯的责任。但是，法官们也一致认为使用Grokster的服务来追踪版权侵犯的行为是违法的。在裁决这个案件时，法官们很谨慎地没有在裁决中使用Sony的先例。相反，他们发现Grokster应该为“诱导”版权侵犯的行为负责，例如他们的广告。但是，法院并没有裁决P2P技术本身冒犯了法律（这正是MGM想要的裁决）。根据高级法院的裁决，Grokster被迫向音像业赔款5亿美金，这足以使Grokster歇业了。

练习

判断练习1～20是对是错：

A. 对　　B. 错

1. 无损压缩意味着恢复的数据没有丢失任何原始信息。
2. 计算机用模拟形式表示信息。
3. 计算机必须使用二进制记数系统表示信息。
4. 数字信号表示任何时间点的两个值之一。
5. 4个二进制位可以表示32种状态。
6. 数字的符号数值表示法有两种表示0的方法。
7. 当为结果分配的位容不下计算出的值时，将发生溢出。
8. 在ASCII字符集中，大写字母和小写字母没有区别。
9. Unicode字符集包括ASCII字符集中的所有字符。
10. 关键字编码是用单个字符代替常用的单词。
11. 行程长度编码适用于压缩英语文本。
12. 赫夫曼编码使用变长的二进制串表示字符。
13. 音频信号的数字化是定期对它进行采样。
14. CD是用二进制格式存储音频信息的。
15. MP3音频格式舍弃了人耳听不到的声音。
16. RGB值用三个数字值表示一种颜色。
17. 索引颜色增加了图像可以使用的颜色数，因此增加了文件的大小。
18. 光栅图形格式只有位图、GIF和JPEG三种。
19. 矢量图形用线段和几何形状表示图像。
20. 时间压缩方法中采用关键帧表示连续帧之间的变化。

从下列清单中，为练习21～26选择正确的术语。

A. 符号数值表示法　　B. 小数点（radix_point）

C. 使用频率　　D. 采样

E. 模拟　　F. 数字

21. ____数据是信息的连续表示法。
22. 从中学时就开始使用的数字表示法是____。
23. 如果数字的基数不是10，我们称小数点（decimal_point）为____。
24. ____数据是信息的离散表示法。
25. 赫夫曼编码是基于字符的____创建的。
26. 音频信号的数字化是定期对它进行____。

练习27～79是问题或简答题。

27. 为什么数据压缩是当前的重要课题？
28. 有损数据压缩和无损数据压缩的区别是什么？
29. 为什么计算机不能处理模拟数据？
30. 具有长秒针的时钟是模拟设备还是数字设备？请解释原因。
31. 将某物数字化是什么意思？
32. 什么是脉冲编码调制？
33. 用下列位数，可以表示多少种状态：
    a) 4位　　b) 5位　　c) 6位　　d) 7位
34. 尽管从二年级开始，你就会计算简单的算术运算，请做下列小测试，确定你是否完全理解了有符号整数的运算。计算下列表达式，其中W等于17，X等于28，Y等于－29，Z等于－13。
    a) X＋Y　　b) X＋W
    c) Z＋W　　d) Y＋Z
    e) W－Z　　f) X－W
    g) Y－W　　h) Z－Y
35. 使用十进制实数直线图验证下列运算的解决方法，其中A等于5，B等于－7。
    a) A＋B　　b) A－B
    c) B＋A　　d) B－A
36. 给定一种定长的数字模式，十进制补码公式中的*k*为6，回答下列问题。
    a) 可以表示多少正整数？
    b) 可以表示多少负整数？
    c) 绘制实数直线图，显示出三个最小的正数和三个最大的正数、三个最小的负数和三个最大的负数以及0。
37. 使用练习36c中绘制的实数直线图，计算下列表达式，其中A等于－499999，B等于3。
    a) A＋B　　b) A－B
    c) B＋A　　d) B－A
38. 使用十进制补码的公式和3.2.1节中描述的模式，计算下列数字。
    a) 35768　　b) －35768
    c) －444455　　d) －123456
39. 在计算练习38中的十进制补码时，是不是遇到很多从0借位的问题？这种计算易于出错。有种窍门可以使这种计算变得容易，而且不会出

错，即让被减数中的每一位都是9，得到的结果再加1。如果被减数的每一位都是9，得到的数字被称为减数的九进制补码。

a) 证明一个数的九进制补码加1等于这个数的十进制补码。

b) 用九进制补码加1的方法计算练习38b、c和d中的值。

c) 你认为哪种方法简单一些，是直接计算十进制补码，还是用九进制补码加1？请给出理由。

40. 使用二进制补码计算下列表达式，其中A等于11111110，B等于00000010。

a) A＋B　　b) A－B

c) B－A　　d) －B

e) －(－A)

41. 一个数的二进制补码一定是负数吗？请解释。

42. 设计一个以11为基数的记数系统。

a) 绘制实数直线图。

b) 展示一个加法示例和一个减法示例。

c) 基于十一进制补码，设计一种负数的表示法。

43. 用算法形式表示符号数值系统中的减法法则。

44. 把下列实数转换成二进制的（5个二进制位）。

a) 0.50　　b) 0.26　　c) 0.10

45. 把下列实数转换成八进制的（5个八进制位）。

a) 0.50　　b) 0.26　　c) 0.10

46. 能够一眼看出八进制小数对应的二进制值，或者看出二进制小数对应的八进制值吗？请解释。

47. 表示包含45个字符的字符集需要多少位？为什么？

48. 把十进制数175.23表示为符号、尾数和指数的形式。

49. ASCII字符集和Unicode字符集间的主要区别是什么？

50. 为几个简单单词创建一个关键字编码表。选择一个段落，用这种编码模式重写这一段。计算压缩率。

51. 下列字符串的行程长度编码是什么？压缩率是多少？

AAAABBBCCCCCCCCDDDD hi there EEEEEEEEEFF

52. 行程长度编码*X5*A9表示什么字符串？

53. 根据下列赫夫曼编码表，译解下列位串。

赫夫曼编码	字　符
00	A
11	E
010	T
0110	C
0111	L
1000	S
1011	R
10010	O
10011	I
101000	N
101001	F
101010	H
101011	D

a) 1101110001011

b) 0110101010100101011111000

c) 101001001010000100010000101001100110

d) 101000100101010001000111010001000011

54. 人是如何接收声音的？

55. 立体声系统的扬声器是模拟设备还是数字设备？请解释。

56. 什么是RGB值？

57. 色深度指什么？

58. 像素分辨率如何影响图像的视觉效果？

59. 请解释时间视频压缩技术。

60. 请描述一种空间视频压缩技术适用的情况。

61. 请定义与数字化声波相关的采样。

62. 高采样率制造的声音质量好，还是低采样率制造的声音质量好？

63. 要再现合理的声音，至少需要多大的采样率？

64. 塑胶唱片和激光唱盘记录声音的方式一样吗？

65. RGB值（130，0，255）是什么意思？

66. RGB值（255，255，255）表示什么颜色？

67. 什么是分辨率？

68. GIF格式采用的是什么技术？

69. GIF文件最适用于什么图像？

70. 各种视频编译码器的相似之处是什么？

71. 它们的不同之处又是什么？

72. 请列出两种视频压缩方法。

73. 我们对到达视网膜的各种频率的光的感觉叫做什么？
74. 表示照片颜色图像的最佳格式是什么？
75. 缩减电影大小的技术是什么？
76. 使应用程序只支持某些特定颜色的技术是什么？
77. 用线段和几何形状刻画图像的格式是什么？
78. 什么格式逐个像素地存储信息？
79. 增强彩色和真彩色之间的区别是什么？

思考题

1. 使用常用（标准）字符集的优点是什么？缺点又是什么？
2. 把整数从一种基数转换成另一种基数的，需要用新基数除这个数。把小数部分从一种基数转换成另一种基数的，则需要用新基数乘这个数。请用位置记数法解释这些算法。
3. 技术在飞快地发展。自从本书面世后，数据压缩技术又发生了哪些变化呢？
4. Napster先例中的争论是MGM对Grokster的案例中争论的核心吗？这两个案件有什么异同？
5. MGM和Grokster达成了什么样的合理协议？你同意法院对这个特殊案件的裁决吗？

第二部分 硬件层

第4章 门和电路

计算机是电子设备，它的大多数基础硬件元件控制着电流。在非常原始的意识中，人类能通过复杂的技术利用电流的能量，再加上我们的意志，就可以进行计算，做出决定。这一章将在计算机科学和电子工程学之间切换，分析计算机中最基础的硬件元件。

在第2章中，我们概括地介绍了记数系统，特别说明了二进制记数系统。从第3章我们了解到，二进制记数系统之所以让人如此感兴趣，因为计算机是采用它表示信息的。在这一章中，我们将探讨计算机如何使用电信号来表示和操作这些二进制值。

目标

学完本章之后，你应该能够：

- 识别基础的门，描述每种门的行为。
- 描述如何用晶体管实现门。
- 用基础门组合成电路。
- 用布尔表达式、真值表和逻辑框图描述门或电路的行为。
- 比较半加器和全加器之间的异同点。
- 描述多路复用器是如何运作的。
- 解释如何操作S-R锁存器。
- 描述四代集成电路的特征。

4.1 计算机和电学

任何电信号都有电压电平。在上一章中提到过，我们根据信号的电压电平区分信号的值(0或1)。一般说来，0～2伏的电压电平是低电压，由数字0表示。2～5伏范围内的信号是高电压，由数字1表示。计算机中的信号被限制在这两个范围之内。

门是对电信号执行基本运算的设备。一个门接受一个或多个输入信号，生成一个输出信号。门的类型很多，在这一章中，我们将分析6种最基本的类型。每种类型的门执行一个特定的逻辑函数。

电路是由门组合而成的，可以执行更加复杂的任务。例如，电路可以用来执行算术运算和存储值。在电路中，一个门的输出值通常会作为另一个门或多个门的输入值。电路中的电流由经过精心设计的相互关联的门逻辑控制。

描述门和电路的表示法有三种，它们互不相同，但却一样有效：

- 布尔表达式

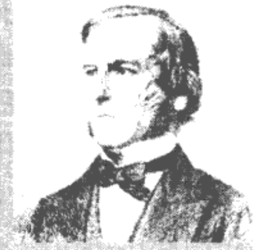
- 逻辑框图
- 真值表

> **门**（gate）：对电信号执行基本运算的设备，接受一个或多个输入信号，生成一个输出信号。
> **电路**（circuit）：相互关联的门的组合，用于实现特定的逻辑函数。

在关于门和电路的讨论中，我们将分析这三种类型的表示法。

英国数学家George Boole发明了一种代数运算，其中变量和函数的值只是0或1。这种代数称为**布尔**（Boolean）**代数**，它的表达式是演示电路活动的极好方式。布尔代数特有的运算和属性使我们能够用数学符号定义和操作电路逻辑。我们在第6至8章讨论高级程序设计语言时还会涉及布尔表达式。

George Boole[1]



布尔代数是以它的发明者英国数学家George Boole的名字命名的。Boole生于1815年，他的父亲是个零售商，在他幼年时就开始教他数学。但最初Boole对古典文学、语言和宗教更感兴趣，这些也是终其一生的兴趣所在。从20岁起，他开始自学法语、德语和意大利语。他精通Aristotle、Spinoza、Cicero和Dante的著作，而且自己也写了几篇哲学论文。

16岁时，他在一所私立学校担任助教，以补贴家用。这份助教工作和另外一份教学工作使他几乎没有时间来学习。几年后，他开办了一所学校，并且开始自学高等数学。虽然缺乏正规的训练，但24岁时，他在《Cambridge Mathematical Journal》上发表了自己的第一篇学术论文。1849年，他被爱尔兰Cork市的Queen学院聘为数学教师。在此，他成为了数学教授，而且余下的职业生涯都是在此度过的。Boole于1864年去世，此时正值他事业的顶峰，在此之前，他发表了50多篇论文，致力于自己的几项主要工作。

Boole的著作《The Mathematical Analysis of Logic》发表于1847年，它最终为数字计算机的开发奠定了基础。在这本书中，Boole阐述了正式的逻辑学公理（与几何学公理类似），建立了数理逻辑这一领域。Boole提出了符号和代数运算来创建他的逻辑学系统。他把1关联到万有集（表示宇宙万物的集合），把0关联到空集，把自己的系统限制在这两个量上，然后定义了减法、加法和乘法的模拟运算。

1854年，Boole发表了《An Investigation of the Laws of Thought, on Which Are Founded the Mathematical Theories of Logic and Probabilities》一书。这本书记述了以他的逻辑学公理为依据的定律，扩展了代数，展示了逻辑系统的计算能力。5年后，Boole发表了《Treatise on Differential Equations》一书，随后又发表了《Treaties on the Calculus of Finite Differences》一书，这本书是处理计算准确性的数值分析的基础著作之一。

Boole的工作并没有为他带来太多的表彰和荣誉。考虑到布尔代数在现代科技中的重要地位，20世纪之前Boole的逻辑系统居然没有受到重视，这令人难以置信。George Boole真正是计算机科学的奠基人之一。

逻辑框图是电路的图形化表示。每种类型的门由一个特定的图形符号表示。通过用不同方法把这些门连接在一起，就可以真实地表示出整个电路逻辑。

真值表列出了一种门可能遇到的所有输入组合和相应的输出，从而定义了这种门的功能。我们可以设计更复杂的真值表，用足够多的行和列说明对任何一套输入值，整个电路如何运作。

> **布尔代数**（Boolean algebra）：表示二值逻辑函数的数学表示法。
>
> **逻辑框图**（logic diagram）：电路的图形化表示，每种类型的门有自己专用的符号。
>
> **真值表**（truth table）：列出了所有可能的输入值和相关的输出值的表。

4.2 门

计算机中的门有时又叫做逻辑门，因为每个门都执行一种逻辑函数。每个门接收一个或多个输入值，生成一个输出值。由于我们处理的是二进制信息，所以每个输入和输出值只能是0（对应低电压信号）或1（对应高电压信号）。门的类型和输入值决定了输出值。

我们将分析下列6种类型的门。在分析完这些门之后，将说明如何把它们组合成电路，执行数学运算。

- 非（NOT）门
- 与（AND）门
- 或（OR）门
- 异或（XOR）门
- 与非（NAND）门
- 或非（NOR）门

在本书中，我们用不同的逻辑框图符号表示每种门，以帮助你理解它们。在分析完整的电路时，这些不同形状的符号可以帮助你分辨不同的门。

> **什么是毫微科学？**
>
> 毫微科学是对小于100毫微米（相当于人的头发宽度的1/100）的材料的研究。科学家预计毫微科学最终将制造出更坚固、更轻、更便宜的材料。两个毫微管（每个宽度相当于10个原子）已经被用于创建一个简单电路。“它们是目前世界上唯一有可能使开关处理信息的速度比最快的硅晶体管还快的东西”，IBM公司的物理科学研究总监Tom Theis这样描述毫微管。
>
> 位于纽约的Rensselaer纳米技术中心的总监Richard W. Siegel说，“如果毫微技术真的有我们所预料的影响，它将像工业革命一样，使社会和工业界都发生重组。”[2]

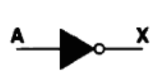
4.2.1 非门

非门接受一个输入值，生成一个输出值。图4-1展示了非门的三种表示方法，即它的布尔表达式、逻辑框图符号和真值表。在这些表示法中，变量A表示输入信号，可以是0或1。变量X表示输出信号，其值可以是0或1，由A的值决定。

布尔表达式

X = A'

逻辑框图符号



真值表

A	X
0	1
1	0

图4-1 非门的各种表示法

根据定义，如果非门的输入值是0，那么输出值是1；如果输入值是1，则输出值是0。非门有时又叫做**逆变器**，因为它将对输入值求逆。

在布尔表达式中，非操作由求反的值之后的'标记表示。有时，也用求反的值上面的单杠表示这个运算。在图4-1中的布尔表达式中，X的值是对输入值A进行求反操作得到的。这是**赋值语句**的一个例子，在赋值语句中，等号左边的变量的值来自于等号右边的表达式。在第6章关于问题解决和算法设计的讨论中将进一步讨论赋值语句。

非门的逻辑框图符号是一个末端具有小圆圈（叫做**求逆泡**）的三角形。输入和输出由流入和流出门的连接线表示。有时，这些连接线上面有标记，但是并非总有标记。

图4-1中的真值表列出了非门所有可能的输入值和对应的输出值。由于非门只有一个输入信号，而且这个信号只能是0或1，所以在真值表中A这一列只有两种可能。X列显示的是非门的输出，即输入值的逆。注意，在这三种表示法中，只有真值表真正定义了非门在各种情况下的行为。

这三种表示方法只是同一事物的不同表示。例如，布尔表达式0'的值总是1。布尔表达式1'的值总是0。这种行为与真值表所示的值一致。

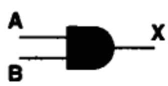
4.2.2 与门

图4-2展示了与门。和非门不同的是，与门接受的输入信号不是一个，而是两个。这两个输入信号的值决定了输出信号。如果与门的两个输入信号都是1，那么输出是1；否则，输出是0。

布尔表达式

$X = A \cdot B$

逻辑框图符号



真值表

A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

图4-2 与门的各种表示法

在布尔代数中，与操作由点（·）表示，有时也表示为星号（*）。该运算符通常可以省略。例如，A · B通常被写作AB。

因为有两个输入值，每个输入有两种可能的值，所以与门的输入可能有四种0和1的组合。因此，在布尔表达式中，用与运算符可能出现四种情况：

$0 \cdot 0 = 0$

$0 \cdot 1 = 0$

$1 \cdot 0 = 0$

$1 \cdot 1 = 1$

同样地，列出与门行为的真值表中有四行，展示了四种可能的组合。真值表的输出列与布尔表达式的结果一致。

4.2.3 或门

图4-3展示了或门。和与门一样，或门也有两个输入。如果这两个输入值都是0，那么输

出是0；否则，输出是1。

布尔表达式

X = A + B

逻辑框图符号



真值表

A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

图4-3 或门的各种表示法

在布尔代数中，或操作由加号（+）表示。或门有两个输入，每个输入有两种可能的值，所以，和与门一样，或门有四种输入组合，在真值表中有四行。

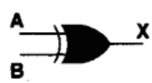
4.2.4 异或门

图4-4展示了异或门。如果异或门的两个输入相同，则输出为0；否则，输出为1。注意异或门和或门之间的区别，只有一种输入使它们的结果不同。当两个输入信号都是1时，或门生成1，而异或门生成0。

布尔表达式

$X = A \oplus B$

逻辑框图符号



真值表

A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

图4-4 异或门的各种表示法

有时，正规的或门又叫做同或门，因为无论一个输入信号是1，还是两个输入信号都是1，它都生成1。而只有当两个输入信号不同时，异或门才生成1。异或门的俗语是“当我说或时，指的是这一个，或者另一个，而不是指它们两个”。

有时，用布尔代数符号⊕表示异或运算，但也可以用其他运算符表示它，我们将此留作练习。

注意，异或门的逻辑框图符号和或门的相似，只是多了一条贯穿两个输入信号的连接线的曲线。

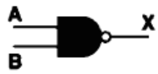
4.2.5 与非门和或非门

图4-5展示了与非门，图4-6展示了或非门。它们都接受两个输入值。与非门和或非门分别是与门和或门的对立门。也就是说，如果让与门的结果经过一个逆变器（非门），得到的输出和与非门的输出一样。

在布尔代数中，通常没有表示与非门和或非门的专用符号，而是根据它们的定义来表示这些概念。也就是说，与非门的布尔表达式是对与运算求逆。同样地，或非门的布尔表达式是对或运算求逆。

布尔表达式 X = (A · B)'

逻辑框图符号



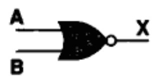
真值表

A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

图4-5 与非门的各种表示法

布尔表达式 X = (A + B)'

逻辑框图符号



真值表

A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

图4-6 或非门的各种表示法

与非门和或非门的逻辑框图符号和与门及或门的相似，只是多了一个求逆泡（说明求逆运算）。比较与门和与非门的真值表中的输出列，它们每一行都是相反的。或门和或非门的真值表也是如此。

4.2.6 门处理回顾

我们已经看过了6种类型的门。要记住它们是如何运作的，看来不是个轻松的任务。其实，这取决于你如何考虑它们。我们绝对不鼓励你去记真值表。这些门的处理都可以用通用的术语简短地描述。如果你这样考虑它们，就可以在需要的时候生成适合的真值表。

让我们回顾一下每种门的处理。有些描述说明了什么样的输入会生成1作为输出，在其他情况下会生成0作为输出。

- 非门将对它的唯一输入值求逆。
- 如果两个输入值都是1，与门将生成1。
- 如果一个输入值是1，或者两个输入值都是1，或门将生成1。
- 如果只有一个输入值是1，而不是两个，异或门将生成1。
- 与非门生成的结果和与门生成的相反。
- 或非门生成的结果和或门生成的相反。

一旦记住了这些一般处理规则，剩下的就是记住布尔运算符和逻辑框图符号。记住，有几种逻辑框图符号只是其他逻辑框图符号的变异。

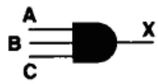
4.2.7 具有更多输入的门

门可以被设计为接受三个或更多个输入值。例如，具有三个输入值的与门，只有当三个输入值都是1时，才生成值为1的输出。具有三个输入值的或门，如果任何一个输入值为1，则生成的输出都是1。这些定义和具有两个输入值的门的定义一致。图4-7展示了具有三个输入信号的与门。

布尔表达式

X = A · B · C

逻辑框图符号



真值表

A	B	C	X
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

图4-7 三输入与门的各种表示法

具有三个输入的门有2^3或8种可能的输入组合。回忆一下，第3章中介绍过，n个不同的输入值有2^n种0和1的组合。这决定了真值表中的行数。

对于逻辑框图符号，只需要在原始符号上加入第三个输入信号即可。但对应布尔表达式，则需要重复一次与操作，以表示第三个值。

4.3 门的构造

在介绍如何把门连接成电路之前，让我们来介绍一些更基础的知识，如何构造门来控制电流。

晶体管

门使用晶体管建立输入值和输出值之间的映射。**晶体管**的角色有两种，一种是传导电流的电线，另一种是阻止电流的电阻器，输入信号的电压电平决定了晶体管的角色。虽然晶体管没有可移动的部分，但是却可以作为开关。它是由**半导体**材料制成的。所谓半导体材料，既不是像铜那样的良好导体，也不是像橡胶一样的绝缘体。通常，使用硅来制造晶体管。

晶体管（transistor）：作为导线或电阻器的设备，由输入信号的电压电平决定它的作用。

半导体（semiconductor）：既不是良好的导体也不是绝缘体的材料，如硅。

在第1章中，我们提到过晶体管的发明（它于1947年诞生在Bell实验室）改变了科技的面貌，开创了计算机硬件的第二个时代。在晶体管之前，数字电路使用的是真空管，这种设备会大量发热，而且常出故障，需要更换。晶体管比真空管小得多，而且运行所需的能量也少。它们可以在几纳秒中转换状态。现在我们知道，计算的发展很大程度上源于晶体管的发明。

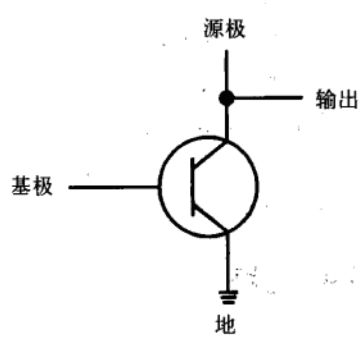


图4-8 晶体管的线路

在分析晶体管的细节之前，我们来讨论一些基本的电学法则。每个电信号都是有源的，如电池或墙上的插座。如果电信号是接地的，那么它可以通过一条备选线路流入地，在地上它是无害的。接地的电信号将被降低或减小到0伏。

晶体管具有三个接线端，即源极、基极和发射极。发射极通常被连接到地线，如图4-8所示。在计算机中，源极制造

的是高电压，约为5伏。基极值控制的门决定了是否把源极接地。如果源极信号接地了，它将被降低到0伏。如果基极没有使源极信号接地，源极信号仍然是高电压。

通常，源极连线上都有一条输出连线。如果源极信号被接地了，那么输出信号是低电压，表示二进制数字0。如果源极信号仍为高电压，那么输出信号也是高电压，表示二进制数字1。

晶体管只能是开（生成高电压输出）或关（生成低电压输出）两种状态，由基极电信号决定。如果基极信号是高电压（接近+5伏），源极信号将被接地，从而关闭了晶体管。如果基极信号是低电压（接近0伏），则源极信号仍然是高电压，晶体管将被打开。

现在，让我们看看如何用晶体管制造各种类型的门。根据晶体管的运作方式，可以证明，最容易创建的门是非门、与非门和或非门。图4-9说明了如何用晶体管构造这些门。

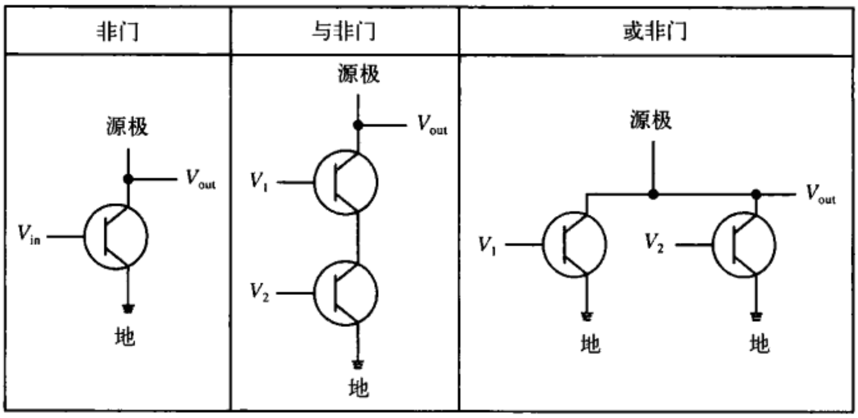


图4-9 用晶体管构造门

非门的图解几乎与原始晶体管的图解一样。它只用一个晶体管制造了一个非门。信号V_{in}表示非门的输入信号。如果它是高电压，那么源极将被接地，输出信号V_{out}是低电压。如果V_{in}是低电压，那么源极不会被接地，V_{out}是高电压。因此，输入信号被逆转了，这正是非门所做的操作。

与非门需要两个晶体管。输入信号V_1和V_2表示与非门的两个输入。如果这两个输入信号都是高电压，那么源极将被接地，输出V_{out}是低电压。但如果有一个输入信号是低电压，那么就会有一个晶体管不使源极信号接地，输出V_{out}是高电压。因此，如果V_1或V_2或二者都是低电压（0），那么输出是1。这和与非门的处理一致。

或非门的构造也需要两个晶体管。同样地，V_1和V_2表示门的输入。但这次晶体管不是串连的。源极分别与每个晶体管连接在一起。如果任何一个晶体管使源极接地了，输出是0。因此，只有当V_1和V_2都是低电压（0）时，输出的才是高电压（1），这和从或非门得到的结果一致。

如我们在这一章的前面所述，与门生成的结果和与非门的完全相对。因此，要构造与门，只需要把与非门的结果传递给一个逆变器（非门）即可。这就是为什么与门比与非门的构造复杂。与门需要三个晶体管，其中两个用于构造与非门，一个用于构造非门。或非门和或门之间存在同样的关系。

4.4 电路

既然我们已经知道单独的门是如何运作的，以及它们的真正构造，那么让我们来看看如何把门组合成电路。电路可以分为两大类。一类是**组合电路**，输入值明确决定了输出。另一类是

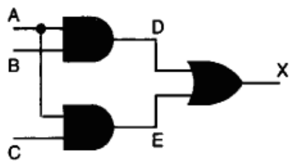
时序电路，它的输出是输入值和电路现有状态的函数。因此，时序电路通常涉及信息存储。我们在本章中介绍的大多数电路都是组合电路，不过也会简短地介绍时序存储器电路。

> **组合电路**（combinational circuit）：输出仅由输入值决定的电路。
> **时序电路**（sequential circuit）：输出是输入值和电路当前状态的函数的电路。

和门一样，我们能用三种方法描述整个电路的运作，即布尔表达式、逻辑框图和真值表。它们是不同的表示法，但却同样有效。

4.4.1 组合电路

把一个门的输出作为另一个门的输入，就可以把门组合成电路。例如，考虑下面的电路逻辑框图：



两个与门的输出被用作或门的输入。注意，A同时是两个与门的输入。图中的连接点说明两条连接线是相连的。如果两条交叉的连接线的交汇处没有连接点，应该看作是一条连接线跨过了另一条，它们互不影响。

这个逻辑框图的意思是什么呢？让我们倒着看看，对于一个特定的结果，它的输入是什么。如果最后的输出X是1，那么D或者E中至少有一个是1。如果D是1，那么A和B必须都是1。如果E是1，那么A和C必须都是1。D和E可以同时为1，但不是必需的。仔细分析这个逻辑框图，确保这种推理和你对门的理解一致。

现在，我们用真值表来表示整个电路的处理：

A	B	C	D	E	X
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	0	0	0
1	0	1	0	1	1
1	1	0	1	0	1
1	1	1	1	1	1

因为这个电路有三个输入，所以需要8行来描述所有可能的输入组合。中间列显示了电路的中间值（D和E）。

最后，让我们用布尔表达式来表示这个电路。因为电路是一组互连的门，所以表示电路的布尔表达式是布尔运算的组合。只需要把这些运算组织成正确的形式，就可以创建一个有效的布尔代数表达式。在这个电路中，有两个与表达式。每个与运算的输出是或运算的输入。因此，下面的布尔表达式表示了这个电路（其中省略了与运算符）：

$$(AB + AC)$$

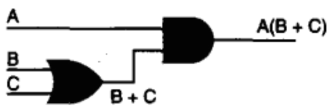
在编写真值表时，用这些布尔表达式标示列比用任意的变量（如D、E和X）好，可以清

楚地标示出这个列表示的是什么。其实，我们也可以用布尔表达式标示逻辑框图，取消图中的中间变量。

现在，让我们从另一个方向入手，从布尔表达式绘制对应的真值表和逻辑框图。考虑下面的布尔表达式：

$$A(B+C)$$

在这个表达式中，两个输入值B和C将进行或运算。这个运算的结果将和A一起，作为与运算的输入，以生成最后的结果。因此，它对应的逻辑框图如下：



同样地，让我们把这个电路表示为真值表。与前面的例子一样，因为这个电路有三个输入值，所以真值表中有8行：

A	B	C	B + C	A(B + C)
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	1	0
1	0	0	0	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

从真值表中任选一行，用逻辑框图验证最后结果是一致的。多试验几行，以便熟悉跟踪电路逻辑的过程。

现在，比较这两个例子中的真值表的最后一列。它们是完全一样的。因此，我们刚才演示了**电路等价**。也就是说，对每个输入值的组合，两个电路都生成完全相同的输出。

电路等价（circuit equivalence）：对应每个输入值组合，两个电路都生成完全相同的输出。

其实，这种现象证明了布尔代数的一个重要属性——**分配律**：

$$A(B+C)=AB+AC$$

这是布尔代数的一大优点，它允许我们利用可证明的数学法则来设计逻辑电路。下表列出了布尔代数的一些属性：

属 性	与	或
交换律	AB = BA	A + B = B + A
结合律	(AB)C = A(BC)	(A + B) + C = A + (B + C)
分配律	A(B + C) = (AB) + (AC)	A + (BC) = (A + B)(A + C)
恒等	A1 = A	A + 0 = A
余式	A(A') = 0	A + (A') = 1
德 · 摩根定律	(AB)' = A' OR B'	(A + B)' = A'B'

这些属性与我们对门处理的理解、真值表和逻辑框图一致。例如，交换律属性，用通俗的话说，就是输入信号的顺序并不重要。（用每个门的真值表来验证它。）余式的意思是，如果把一个信号和它的逆作为与门的输入，那么得到的一定是0，但如果把一个信号和它的逆作为或门的输入，那么得到的一定是1。

在布尔代数中，有一个非常著名也非常有用的定律——德·摩根定律。这个定律声明，对两个变量的与操作的结果进行非操作，等于对每个变量进行非操作后再对它们进行或操作。也就是说，对与门的输出求逆，等价于先对每个信号求逆，然后再把它们传入或门：

$$(AB)' = A' + B'$$

这个定律的第二部分是，对两个变量的或操作的结果进行非操作，等于对每个变量进行非操作后再对它们进行与操作。用电路术语来说，就是对或门的输出求逆，等价于先对每个信号求逆，然后再把它们传入与门：

$$(A + B)' = A'B'$$

德·摩根定律和其他布尔代数属性为定义、管理和评估逻辑电路的设计提供了正规的机制。

以Augustus DeMorgan命名的德·摩根定律

与George Boole同时代的DeMorgan是1828年伦敦大学的第一位数学教授，他在此执教了30年。他编写了关于算术、代数、三角法和微积分学的基础课本，发表过关于建立逻辑计算的可能性和用符号表示想法的基本问题的论文。虽然DeMorgan不是德·摩根定律的发现者，但是他正式陈述了这一定律，也就是今天我们所见到的。[3]

4.4.2 加法器

计算机能执行的最基本运算可能就是把两个数相加。在数字逻辑层，加法是用二进制执行的。第2章讨论了这一过程。这些加法运算是由专用电路**加法器**执行的。

与所有记数系统中的加法一样，对两个二进制数求和的结果可能生成进位值。例如，在二进制中，1 + 1 = 10。计算两个数位的和并生成正确进位的电路叫做**半加器**。

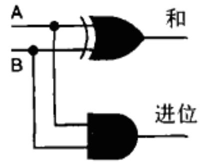
加法器（adder）：对二进制值执行加法运算的电路。
半加器（half adder）：计算两个数位的和并生成正确进位的电路。

让我们看看求两个二进制数字A与B的和的所有可能。如果A和B都是0，那么和为0，进位为0。如果A是0，B是1，则和为1，进位为0。如果A是1，B是0，则和为1，进位为0。如果A和B都是1，那么和为0，进位为1。相应的真值表如下：

A	B	和	进位
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

实际上我们计算的是两个输出结果，即和与进位。所以电路有两条输出线。

如果把和与进位列同各种门的输出比较，你会发现，和对应的是异或门，进位对应的是与门。因此，下列逻辑框图表示了半加器：



用各种输入值组合测试这个框图，确定它生成的两个输出值是什么。结果符合二进制算术的法则吗？它们应该是符合的。现在，拿你的结果与真值表比较，它们也应该是匹配的。

这个电路的布尔表达式是什么呢？因为这个电路生成两个输出值，所以我们用两个布尔表达式表示它：

$$\text{和} = A \oplus B$$
$$\text{进位} = AB$$

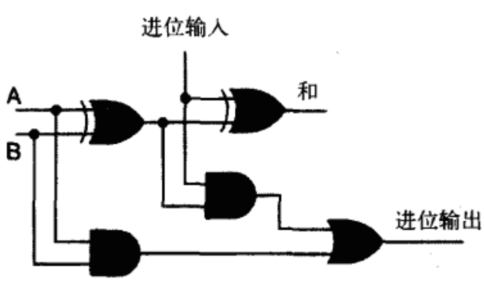
注意，半加器不会把进位（进位输入）考虑在计算之内，所以半加器只能计算两个数位的和，而不能计算两个多位二进制值的和。考虑进位输入值的电路叫做**全加器**。

> **全加器**（full adder）：计算两个数位的和，并考虑进位输入的电路。

可以用两个半加器构造一个全加器。如何做呢？求和的输入必须是进位输入与两个输入值的和。也就是说，把从半加器得到的和与进位输入相加。两个加法都具有进位输出。那么，有可能出现两个进位输出都是1而需要进一步进位吗？幸运地是，不可能出现这种情况。看看半加器的真值表，没有和与进位都是1的情况。

图4-10展示了全加器的逻辑框图和真值表。这个电路有三个输入，即原始的数位A和B以及进位输入值。因此，真值表具有8行。我们将相应的布尔表达式留作练习。

逻辑框图



真值表

A	B	进位输入	和	进位输出
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

图4-10　全加器

要把两个八位值相加，需要复制8次全加器电路。一个位值的进位输出将用作下一个位值的进位输入。最右边的位的进位输入是0，最左边的位的进位输出将被舍弃（通常会生成溢出错误）。

改进这些加法器电路设计的方法有很多，但在本文中我们不再探讨它们的细节。

4.4.3 多路复用器

多路复用器是生成单个输出信号的通用电路。输出值等于该电路的多个输入值之一。多路复用器根据称为选择信号或选择控制线的输入信号选择用哪个输入信号作为输出信号。

> **多路复用器**（multiplexer）：使用一些输入控制信号决定用哪条输入数据线发送输出信号的电路。

让我们看一个例子。图4-11是一个多路复用器的框图。控制线S0、S1和S2决定了用另外8条输入线（D0到D7）中的哪一条发送输出信号（F）。

三条控制线的值将被译为一个二进制数，决定了发送输出信号的输入线。回忆一下，第2章中介绍过，3位二进制数字可以表示8个不同的值，即000、001、010、011、100、101、110和111。注意，这些值只能从0数到7，对应了输出值D0到D7。因此，如果S0、S1和S2都是0，那么多路复用器的输出就是D0。如果S0是1，S1是0，S2是1，那么输出就是D5。

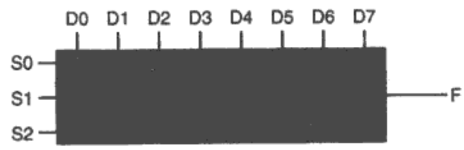


图4-11 具有三条选择控制线的多路复用器的框图

下面的真值表列出了输入控制线如何决定这个多路复用器的输出：

S0	S1	S2	F
0	0	0	D0
0	0	1	D1
0	1	0	D2
0	1	1	D3
1	0	0	D4
1	0	1	D5
1	1	0	D6
1	1	1	D7

图4-11中的框图隐藏了执行多路复用器逻辑的复杂电路。用8个三输入与门和一个8输入或门可以表示这个电路。本书不再详述该电路。

多路复用器可以有任意多条输入线和相应的控制线。一般说来，n条输入控制线的二进制值决定了选择2^n条数据线中的哪一条作为输出。

多路分配器是执行相反操作的电路。也就是说，它只有一个输入，根据n条控制线的值，这个输入信号将被发送到2^n个输出。

4.5 存储器电路

数字电路的另一个重要作用是可以用来存储信息。这些电路构成了时序电路，因为这种电路的输出信号也被用作电路的输入信号。也就是说，电路的下一个状态部分是由当前状态决定的。

存储器电路有很多种。本书只分析一种存储器电路——S-R锁存器。一个S-R锁存器存储一个二进制数字（1或0）。用不同的门，可以以不同的方式设计S-R锁存器。图4-12展示了一种用与非门设计的S-R锁存器。

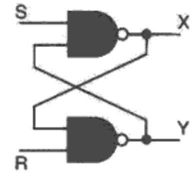


图4-12 一种S-R锁存器

这个电路的设计使两个输出X和Y总是互补的。也就是说，当X是0时，Y是1，反之亦然。X在任意时间点的值都被看作电路的当前状态。因此，如果X是1，电路存储的就是1；如果X是0，电路存储的就是0。

回忆一下，只有当两个输入值都是1时，与非门才会输出0。这个电路中的每个门都有一个外部输入（S或R）和一个来自其他门的输出的输入。假设电路的当前状态存储的是1（也就是说，X是1），S和R也都是1。那么Y仍为0，X仍为1。再假设电路的当前状态存储的是0（X是0），R和S还是1。那么Y仍为1，X仍为0。因此，无论当前存储的值是什么，如果S和R都是1，电路就保持为当前状态。

这个解释说明，只要S和R都是1，S-R锁存器就保留它的值。那么最初如何把一个值存入S-R锁存器呢？暂时把S设置为0，保持R为1，可以把S-R锁存器设置为1。如果S是0，X将变为1。只要S立刻恢复为1，S-R锁存器将保持1的状态。暂时把R设置为0，保持S为1，可以把S-R锁存器设置为0。如果R是0，Y将变为1，因此X也变为0。只要R立刻恢复为1，电路将保持0的状态。

因此，小心控制S和R的值，电路就可以存储0或1。把这个思想扩展至较大的电路，就可以设计出容量较大的存储器设备。

4.6 集成电路

集成电路（又称芯片）是嵌入了多个门的硅片。这些硅片被封装在塑料或陶瓷中，边缘有引脚，可以焊接在电路板上或插入适合的插座中。每个引脚连接着一个门的输入或输出，或者连接着电源，或者接地。

> **集成电路**（integrated circuit）：又称**芯片**（chip），是嵌入了多个门的硅片。

集成电路（IC）是根据它们包含的门数分类的。这些分类也反映了IC技术的发展历史。

缩　写	名　称	门 数 量
SSI	小规模集成	1～10
MSI	中规模集成	10～100
LSI	大规模集成	100～100 000
VLSI	超大规模集成	多于100 000

一个SSI芯片只有几个独立的门，图4-13展示了一种SSI芯片。这个芯片有14个引脚，其

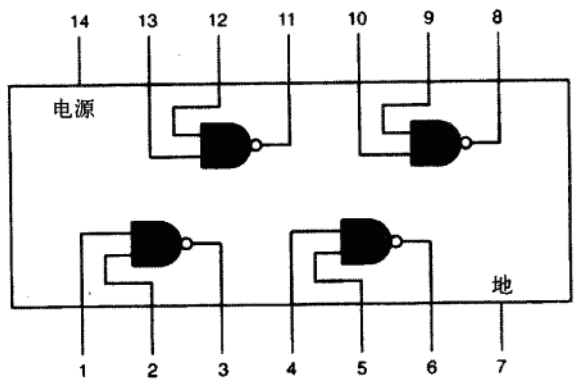


图4-13　包含独立的与非门的SSI芯片

中8个用作门的输入，4个用作门的输出，1个接地，1个接电源。用不同的门可以制成类似的芯片。

一个芯片如何容纳多于100 000个的门呢？那样意味着需要300 000个引脚。答案是VLSI芯片上的门不像小规模集成电路中的门一样，它们不是独立的。VLSI芯片上嵌入的电路具有很高的门引脚比。也就是说，许多门被组合在一起，创建的复杂电路只需要很少的输入和输出值。多路复用器是这种电路的一个例子。

4.7 CPU芯片

计算机中最重要的集成电路莫过于中央处理器（CPU）。下一章会讨论CPU的处理，此刻只要认识到，CPU只是一种具有输入线和输出线的高级电路。

每个CPU芯片都有大量的引脚，计算机系统的所有通信都是通过这些引脚完成的。这些通信把CPU和本身也是高级电路的存储器与I/O设备连接在一起。

关于CPU的处理和它与其他设备之间的交互属于计算机处理的另一个分层，有时这个分层被称为*构件体系结构*。尽管计算机构件体系结构的重点仍然在硬件，但它也应用了抽象法则，使我们能够暂时忽略门和电路这些细节，向完整地理解计算机处理跨进了一步。

虚拟慈善事件

步行马拉松、自行车马拉松和慈善赛跑这样的活动越来越多。负责筹集资金的组织常常会在主办城市造成各种问题，例如交通堵塞，导致人们不能到家或公司，所以一些组织想出了新方案，即虚拟步行或赛跑。例如，Denver的Susan G. Komen Race for the Cure就吸引了60 000多名参与者，它的一个选择就是“在家为医疗筹款”。选择这个选项的参与者可以把筹款请求用电子邮件发送给朋友，而不必真正去赛跑。虚拟事件减少了主办城市的交通堵塞，并且组织费用更少，所以最后得到的筹款更多。

小结

这一章我们讨论了计算机如何通过控制最底层的电流进行运算。由于我们讨论的是使用二进制信息的数字计算机，所以我们只关注两个电压范围，分别表示为二进制数字1或0。电流由称为门的电子设备操纵，门负责执行基本的逻辑运算，如非运算、与运算和或运算。门是由一个或多个晶体管创建的，晶体管的发明使计算学发生了翻天覆地的变化。

把一个门的输出作为另一个门的输入，可以把门组合成电路。仔细设计这些电路，可以创建出能执行更复杂任务（如求和、多路复用和存储数据）的设备。门的集合，或者说完整的电路，常常被嵌入在一个集成电路（或芯片）中，这引出了中央处理器的概念。

道德问题：电子邮件隐私权

你曾经用电子邮件写过重要的消息、发送过简历或者抱怨过你的室友吗？如果你知道陌生人、管理员或者你的室友会看到这些消息，会改用其他方式吗？今天，电子邮件这种曾经只在计算机领域使用的工具，已经成为数百万人通信的标准方式。但是，许多用户都错误地假设只有收信人才能读到电子邮件中的内容。有了这种对隐私权的假想，人们会用电子邮件发送绝对不想让其他人读到的个人信件，或者发送绝密信息，这些信息一旦落入不道德的人手中，就会泄漏出去。电子邮件从

发件人到收件人的途中，将从一个服务器转移到另一个服务器，要阅读电子邮件中的内容，比看明信片还要容易。电子邮件的安全性成了众人争论的焦点，这些争论是为了找出个人权利、组织权利和计算机技术的交集。

许多依靠电子邮件通信的公司现在都草拟了一些政策，规定在哪里终止电子邮件隐私权，在哪里开始电子邮件监控。支持电子邮件监控的人认为，通过公司服务器接收的电子邮件都属于公司，因此公司有权随意访问这些电子邮件。他们辩论说监控能够防止雇员滥用自己的电子邮件使用权，强化了雇主对可能影响到公司的信件控制。反对者则解释说电子邮件监控产生了不信任、不尊重的氛围，为雇员自治设置了不必要的障碍。

围绕电子邮件的隐私权问题已经超出了公司政策的范围。例如，2000年7月，英国通过了研究权利规章的议案，赋予了政府访问所有Internet信件的权利。Internet服务提供商必须把所有电子邮件发送到政府总部，政府官员能够访问用来保护电子邮件的各种加密码。

即使在电子邮件到达了目的地后，也可能被收件人之外的读者看到。大多数电子邮件服务提供的转发功能赋予了收件人在未经作者许可的情况下把电子邮件发送给其他人的权利。研究表明，人们认为读别人的普通邮件是侵犯隐私权，而读电子邮件却不是。这种想法以及偷看和监控电子邮件的做法危害到了电子邮件的安全性。

练习

判断练习1～17是对是错：

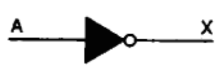
A. 对　　　　B. 错

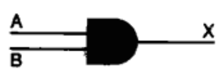
1. 逻辑框图和真值表在表达门和电路的处理方面同样有效。
2. 在表达门和电路的处理方面，布尔表达式比逻辑框图更有效。
3. 非门接受两个输入。
4. 当两个输入都是1时，与门的输出值为1。
5. 对于相同的输入，与门和或门生成的结果相反。
6. 当两个输入都是1时，或门的输出值为1。
7. 当一个输入是0，另一个输入是1时，或门的输出是0。
8. 只有当两个输入都是0时，异或门的输出值才是1。
9. 或非门生成的结果与异或门的结果相反。
10. 一个门可以被设计为接受多个输入的。
11. 晶体管是由半导体材料制成的。
12. 对与门的结果求逆，等价于先分别对输入信号求逆，然后再把它们传递给或门。
13. 两个二进制数字的和（忽略进位）是由与门表示的。
14. 全加器会把进位输入的值计算在内。
15. 多路复用器是把输入线中的所有位相加生成输出的。
16. 集成电路是根据它们包含的门数分类的。
17. CPU是一种集成电路。

为练习18～29中的运算描述或框图选择匹配的门。

A. 与门　　　　B. 与非门
C. 异或门　　　　D. 或门
E. 或非门　　　　F. 非门

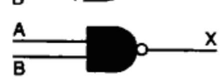
18. 对输入求逆。
19. 只有当所有输入都是1时才生成1，否则生成0。
20. 只有当所有输入都是0时才生成0，否则生成1。
21. 只有当输入相同时才生成0，否则生成1。
22. 如果所有输入都是1，生成0，否则生成1。
23. 如果所有输入都是0，生成1，否则生成0。

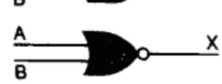
24. 

25. 

26. 

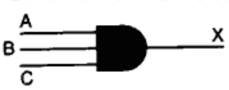
27. 

28. 

29. 

练习30～73是简答题或设计题。

30. 如何用电压电平决定表示电压的二进制数字？
31. 请区分门和电路。
32. 描述门和电路行为的三种表示法是什么？
33. 请分别描述练习32提到的表示法。
34. 一个门可以接受多少个输入信号，可以生成多少个输出信号？
35. 请分别描述6种类型的门。
36. 给出非门的三种表示法，简单明了地说出非的意思。
37. 给出与门的三种表示法，简单明了地说出与的意思。
38. 给出或门的三种表示法，简单明了地说出或的意思。
39. 给出异或门的三种表示法，简单明了地说出异或的意思。
40. 给出与非门的三种表示法，简单明了地说出与非的意思。
41. 给出或非门的三种表示法，简单明了地说出或非的意思。
42. 对比与门和与非门的异同。
43. 给出三输入的与门的布尔表达式，为它做好标记，然后列出它的真值表。



44. 给出三输入的或门的布尔表达式，为它做好标记，然后列出它的真值表。



45. 门用什么建立输入值和输出值之间的映射。
46. 晶体管的行为是什么？
47. 晶体管是什么制成的？
48. 当电信号接地后会出现什么情况？
49. 晶体管的三个接线端是什么，它们是如何操作的？
50. 下列每种门需要多少个晶体管？

 a) 非门 b) 与门 c) 或非门
 d) 或门 e) 异或门

51. 绘制与门的晶体管框图，并解释它的处理。
52. 绘制或门的晶体管框图，并解释它的处理。
53. 如何把门组合成电路？
54. 电路的两大分类是什么，它们有什么不同？
55. 绘制与下列布尔表达式对应的电路图：

 $(A+B)(B+C)$

56. 绘制与下列布尔表达式对应的电路图：

 $(AB+C)D$

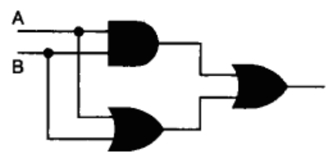
57. 绘制与下列布尔表达式对应的电路图：

 $A'B+(B+C)'$

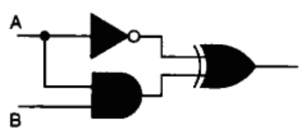
58. 绘制与下列布尔表达式对应的电路图：

 $(AB)'+(CD)'$

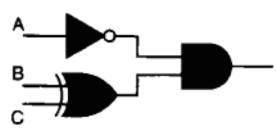
59. 用真值表描述下列电路的行为：



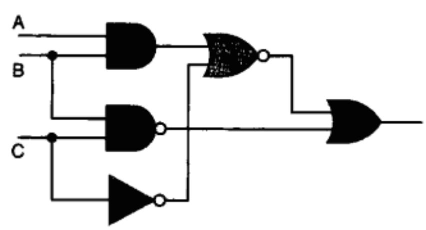
60. 用真值表描述下列电路的行为：



61. 用真值表描述下列电路的行为：



62. 用真值表描述下列电路的行为：



63. 什么是电路等价？
64. 描述布尔代数的6种属性，并解释每种属性的含义。
65. 请区分半加器和全加器。
66. 全加器的布尔表达式是什么？
67. 什么是多路复用器？
68. a) 存储器使用的是什么类型的电路？

 b) S-R锁存器可以存储多少位？

 c) 图4-12中的S-R锁存器设计得到的输出X和Y

是什么？

69. 什么是集成电路或芯片？

70. 定义缩写SSI、MSI、LSI和VLSI。

71. 在图4-13所示的芯片中，引脚的作用是什么？

72. 绘制一个电路，用两个全加器求两个两位二进制数值的和。

73. 用其他运算符如何表示异或运算？

思考题

1. 本章用布尔表达式、真值表和逻辑框图表示同样的门或电路行为。你清楚这三种表示法之间的关系吗？你认为哪种方法最直观，哪种方法最不直观？
2. 有许多情况都可以用本章中的思想描述，例如，单个电灯开关的操作或具有两个开关的电灯的操作。你可以想出日常生活中有哪些情况可以用本章中的方法表示吗？
3. 你曾经遇到过这些情况吗？给某人发送了电子邮件之后立刻就后悔了，或者在电子邮件中写出了自己从来都不会讲的话。请思考这种假定："电子邮件降低了个人言论的礼貌程度"，你同意这种观点吗？
4. 如果某人从一台学校的计算机或商业计算机上发送了一封电子邮件，那么这条消息应该被看作是隐私吗？拥有这台计算机的组织或个人有权利审查这条消息吗？
5. 你认为读别人的普通邮件是侵犯隐私权，而读电子邮件却不是吗？

第5章 计算部件

第2章介绍了计算机表示所有信息采用的二进制记数系统。第4章介绍了如何控制底层电流来管理二进制数值。接下来我们将介绍这些技术利用的主要计算机部件。这些部件就像Lego拼装玩具的组件，Lego的组件能够构造出各种各样的建筑，计算部件则可以组合成各种各样的计算机。

尽管这些部件（如内存和CPU）常常被看作计算机最基本的组成部分，但是我们明白，它们是更加基本的概念的抽象。

目标

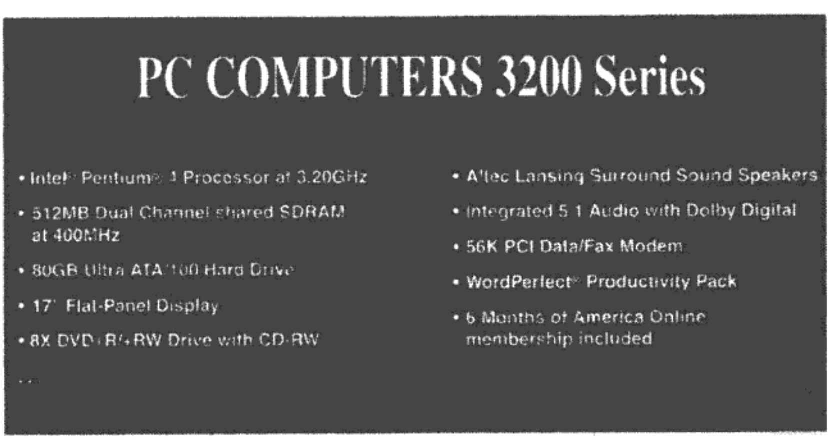
学完本章之后，你应该能够：

- 读懂一则计算机广告，明白其中的行话。
- 列出冯·诺伊曼机的部件和它们的功能。
- 描述冯·诺伊曼机的读取–译解–执行周期。
- 描述如何组织和访问计算机内存。
- 列出并描述不同的辅助存储设备。
- 定义三种并行计算机的配置。

5.1 独立的计算机部件

计算学的专用术语和缩写比大多数领域都多。我们通过翻译一则台式电脑的广告开始本章的介绍。然后，在详细研究每个部件之前，我们将整体介绍一下计算机的部件。

请看下面一则台式电脑的广告：



这则广告有两点既重要，又有趣。首先，一般人看了它，会顿觉迷惑，完全不知所云。再者，其中介绍的机型早已过时。这一章将尽可能地解释各种缩写，但对计算机硬件和软件的更新速度，我们无能为力。

在抽象地介绍计算机部件之前，让我们先来仔细看看这则广告，解释一下其中的缩写。之后，我们将重新深入地介绍前面提到过的各种资料，因此，如果某些术语有些费解，不必担心，它们都将被重新定义。

第一行介绍了计算机的中央处理器，告诉你这台计算机有多么强大。它的中央处理器是Intel Pentium 4。上一章提到过，CPU通常是32位的处理器。3.20GHz说明了处理器有多快。G是giga的缩写，这里表示10^9。Hz代表赫兹，即频率单位，等于每秒一圈，是以Heinrich R. Hertz的名字命名的。因此，这个处理器的速度是每秒3 200 000 000圈（越快越好）。

接下来的一行描述了计算机具有的随机存取存储器（Random Access Memory）——主存储器。这台计算机配备的是512MB（或512×2^{20}字节）的主存。SDRAM是同步动态随机存取存储器的缩写。随机存取表示能够直接存取存储器的每个字节，而不是必须从头开始，顺次存取存储器中的每个字节，直到找到想要的字节为止。400MHz说明能够以每秒400 000 000圈的速度访问存储器。Microsoft公司的CEO比尔·盖茨在1981年曾说过“640K内存对每个人来说都应该够用了”，[1]这一言论被证明是错误的（越大越好）。

《The Wall Street Journal》的专栏作家Walt Mossberg指出了有关处理器速度和内存的一个有趣观点：[2]要提高任务的性能，并不意味着一定要提高处理器的速度，只要拥有足够的内存就可以，这些任务包括网上冲浪、电子邮件和文字处理等。简而言之，内存越大，所需要的处理器的性能越低。不过要注意，用户能够使用的内存并不一定是计算机的所有内存。一些低价的机器可能会用一部分内存提高视频处理的能力。

下面一行描述了硬盘驱动器，即作为二级（辅助）存储设备的磁盘的普通名称。硬盘驱动器安装在机箱中，除了显示器、键盘和鼠标外，其他所有设备都位于机箱中。80GB是存储的字节量（G表示giga，等于2^{30}），因此，这台机器的存储量是80×2^{30}字节（越大越好）。Ultra ATA-100说明了硬盘驱动器使用的数据传输接口的类型，也称为IDE或集成磁盘驱动电路接口。这台机器的硬盘传输数据的速率是每秒100MB（越快越好）。

接下来一行描述的是显示器。平板是显示器的类型，17"是屏幕的对角线长度。虽然广告中没有显示屏幕的可视大小，但是脚注中说明了可视部分为16.0"，像点间距0.28。像点间距指的是屏幕上两个点之间的距离（越小越好）。下一行介绍的是提高机器图形处理能力的专用显卡。

这台机器具备一个DVD-ROM。DVD-ROM是Digital Versatile Disk, Read-Only Memory（数字化视频光盘，只读内存）的缩写。只读表示只能读取驱动器中的磁盘的数据，而不能更改磁盘上的信息。DVD看起来和音乐CD一样。只要有DVD驱动器，就可以在计算机上播放CD，或者观看DVD影片了。2X是访问驱动器上的信息的速度。R/RW是Record（录制）和Read/Write（读/写）的缩写。使用这种驱动器，还可以烧录自己的DVD或CD。这台计算机可以把各种数据、音频或视频文件烧录到CD-ROM或DVD-ROM上。

这台机器还额外附加了CD-RW驱动器，这是一种CD磁盘驱动器，除了读取磁盘上的信息，还可以把信息写入磁盘。

接下来的两行介绍了这台计算机中安装的音响系统。如果你对计算机音乐感兴趣，声卡和扬声器都非常重要。通常计算机都有一个内置的小扬声器，用于发出基本的声音。

下面介绍的是调制解调器，即让你能够连接到Internet的设备。56K说明这种调制解调器每秒能处理56 000个字节。（K表示kilo，即10^3）。下载和上载的速度会根据厂商和连接线的情

况有所不同。下载意思是把Internet上的信息传输到你的机器上，上载意思是把你的计算机上的信息传输到Internet上。

下面的两行介绍了随机附带的软件和授权书。WordPerfect™ Productivity Pack是Corel公司开发的一套软件，包括文字处理软件、电子制表软件、联系人管理程序和照片编辑器。最后一行声明Internet服务提供商AOL将提供半年的Internet访问服务。

这个广告中用到了3种大小度量。让我们概括一下在计算中常用的前缀。

10的幂	2的幂	2的幂的等价值	前　缀	缩　写	词　源
10^{-12}			pico	p	西班牙语中的很少的
10^{-9}			nano	n	希腊语中的矮小的
10^{-6}			micro	μ	希腊语中的小的
10^{-3}			milli	m	拉丁语中的一千
10^{3}	2^{10}	1024	kilo	K	希腊语中的第一千的
10^{6}	2^{20}	1 048 576	mega	M	希腊语中的大的
10^{9}	2^{30}	1 073 741 824	giga	G	希腊语中的巨大的
10^{12}	2^{40}	空间不够	tera	T	希腊语中的庞然大物
10^{15}	2^{50}	空间不够	peta	P	希腊语中5的前缀

你注意到在引用存储量时使用了2的幂，而引用存储时间时使用了10的幂吗？时间是由多个位表示的，因此可以用我们所熟悉的十进制表示。存储量总是以字节表示的，即2的幂，因此应该用2的幂表示它。如果记住这个区别，就会很清楚，表示速度时，K等于1000，表示存储量时，K等于1024。

计算机行话中的大小解析

Admiral Grace Murray Hopper用一卷1000英尺长的线、一小段相当于前臂长短的线和一袋胡椒粒来比喻计算机行话中的相对大小。她指出，线卷是一个电子在一微秒内传输的距离，一小段线是电子在一纳秒内传输的距离，胡椒粒表示电子在一皮秒内传输的距离。她劝告听众要记住自己的每一纳秒。

现在，我们从特例转移到一般。下一节介绍的不再是某个特定计算机的配置，而是从逻辑层构成计算机的各个硬件。

5.2 存储程序的概念

1944至1945年间实现了数据和操作数据的指令的逻辑一致性，而且它们能存储在一起，这是计算历史上的一个主要定义点。这个原理就是著名的*冯·诺伊曼体系结构*，基于这个原理的计算机设计仍然是当前计算机的基础。尽管这个名字把荣誉给了从事原子弹制造的天才数学家John von Neumann，但是这种思想则可能源自J. Presper Eckert和John Mauchly，与von Neumann同时期的两位先驱，他们在宾夕法尼亚大学的Moore学院致力于ENIAC的开发。

谁是现代计算机之父重要吗？

毫无疑问，在20世纪30年代末期和40年代致力于电子计算设备研发的所有人都对我们所熟知的计算机做出了贡献。这张清单上除了大名鼎鼎的von Neumann、Eckert和Mauchly，还有John

Atanasoff、Clifford Berry和Konrad Zuse。

1951年，Sperry Rand买下了ENIAC和它的基本概念的专利，开始向其他计算机制造商收取版税。由于不想支付版税，Honeywell公司研究了现代计算机的历史，提出了证据，证明艾奥瓦州立大学的John Atanasoff的工作直接影响了Mauchly和Eckert。这些证据使ENIAC的专利权于1973年被吊销了。

5.2.1 冯·诺伊曼体系结构

冯·诺伊曼体系结构的另一个主要特征是处理信息的部件独立于存储信息的部件。这一特征导致了下列5个冯·诺伊曼体系结构的部件，如图5-1所示。

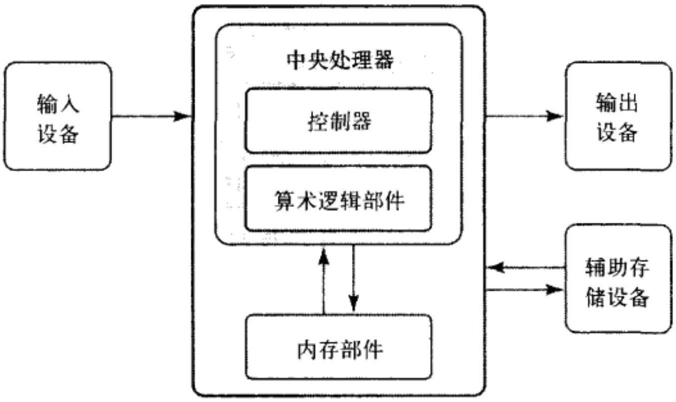


图5-1 冯·诺伊曼体系结构

- 存放数据和指令的内存部件
- 对数据执行算术和逻辑运算的算术逻辑部件
- 把数据从外部世界转移到计算机中的输入部件
- 把结果从计算机内部转移到外部世界的输出部件
- 担当舞台监督，确保其他部件都参与了表演的控制器

John Vincent Atanasoff



John Vincent Atanasoff于1903年10月4日出生在纽约的Hamilton，他是家中的9个孩子之一。在大约10岁时，他的父亲买了一把计算尺。读完说明书后，John Vincent对其爱不释手，并对其涉及的数学知识产生了浓厚的兴趣。他的母亲发现了他的这一兴趣，开始帮助他学习父亲用过的数学课本。他延续了自己对数学和科学的兴趣，仅用了两年时间，就从中学毕业了。之后，他的家移居到了佛罗里达的Old Chicara，John Vincent于1925年获得了佛罗里达大学的电气工程学位，当时，这所大学还不提供理论物理的学位。一年之后，他获得了艾奥瓦州立大学的数学硕士学位。1930年，在获得理论物理的博士学位后，他返回艾奥瓦州立大学，开始担任数学和物理的助理教授。

Atanasoff博士和他的研究生们当时需要进行复杂的数学运算，于是他对发明一种能够进行这些运算的机器产生了兴趣。他研究了当时已有的计算设备，包括Monroe计算器

和IBM的制表机，得出这些机器太慢且不精确的结论，并陷入了寻找解决方案的迷惘中。据他所述，某个夜晚，当他在一个小酒馆喝过一杯威士忌后，他得到了如何构造这种计算设备的想法。这是一种电子设备，能够直接进行逻辑运算，而不是像模拟设备那样，需要列举。它使用的是二进制数，而不是十进制数，内存使用电容器，用再生处理避免漏电造成的失误。

1939年，Atanasoff博士从学校获得了一笔650美元的科研费，在新的助教Clifford Berry的帮助下，他开始在物理大楼的地下室研制Atanasoff Berry计算机（ABC）的第一个样机并在当年研发成功。

1941年，Ursinus学院的物理学家John Mauchly来艾奥瓦州拜访Atanasoff，他是Atanasoff博士在一个会议上结识的。在参观过ABC的样机后，他们进行了长时间的讨论，Mauchly离开时带走了一些介绍ABC设计的论文。此后，Mauchly和J. Presper Eckert在宾夕法尼亚大学的Moore学院继续从事计算设备的研发工作。他们于1945年完成的ENIAC样机成了著名的第一台计算机。

1942年，Atanasoff博士去了华盛顿，在Naval Ordnance实验室担任水下声测量项目的主任，把申请ABC计算机专利的问题交给了艾奥瓦州的律师们。这些律师根本没有提交这项专利申请，最后，在没有通知Atanasoff和Berry的情况下，ABC被拆除了。世界大战结束后，Atanasoff博士担任陆军的首席科学家，以及Naval Ordnance实验室的Navy Fuse项目的指导。

1952年，Atanasoff博士创建了Ordnance Engineering公司，这是一个研究和设计公司，之后被Aerojet General公司收购了。Atanasoff在Aerojet公司继续工作，直到1961年退休。

其间，Mauchly和Eckert于1947年申请了ENIAC计算机的专利。Sperry Rand买下了这个专利，开始收取版税。由于Honeywell公司不愿意支付版税，Sperry Rand诉诸法律。随后的审判工作持续了135个工作日，收集了77个证人（包括Atanasoff博士）的证词，编订成了两万多页的记录。法官Larson裁决Mauchly和Eckert“不是第一个发明自动电子数字计算机的人，他们的机器是从John Vincent Atanasoff博士的机器派生出的”。

1990年，美国总统George Bush授予Atanasoff博士国家科技奖章，以感谢他的先驱性工作。Atanasoff博士逝世于1995年6月15日。

内存

回忆一下关于数制系统的讨论，每个存储单元（称为位）能存放1或0，这些位被组合成字节（8位），字节被组合成字。内存是存储单元的集合，每个存储单元有一个唯一的物理地址。这里用通称*单元*，而不是用字节或字，因为不同机器中每个可编址的位置的位数（称为**可编址性**）不同。目前大多数计算机都是字节可编址的。

> **可编址性**（addressability）：内存中每个可编址位置存储的位数。

上一节中的广告说明了内存是512×2^{20}字节的。也就是说，每512MB是唯一可编址的。因此，这台机器的可编址性是8位。尽管它有4 294 967 296字节的内存，但是只有512MB可用。这个内存字节数是怎么来的呢？它就是2^{32}。广告中提到的Pentium 4处理器是32位的机器。也就是说，处理器能够识别2^{32}个不同的内存地址。请注意处理器位数和地址数之间的关系，*n*位处理器可以产生2^n个不同的地址。

内存的存储单元是从0开始连续编号的。例如，如果可编址性是8位，那么内存就有256个存储单元。存储单元的编号如下：

地 址	内 容
00000000	11100011
00000001	10101001
⋮	⋮
11111100	00000000
11111101	11111111
11111110	10101010
11111111	00110011

地址为11111110的存储单元中的内容是什么？存储在这个位置的位组合是10101010。这是什么意思呢？我们不能抽象地回答这个问题。11111110这个存储单元中存放的是指令？是符号？是二进制补码？还是图像的一部分？由于不知道这个内容表示的是什么，我们不能确定它的意思。它只是一个位组合。要确定位组合表示的信息，必须给它们一个解释。

在提到字节或字中的位时，都是从0开始，从右到左对位编号的。上述地址11111110中的位是如下编号的：

7	6	5	4	3	2	1	0	← 位位置
1	0	1	0	1	0	1	0	← 内容

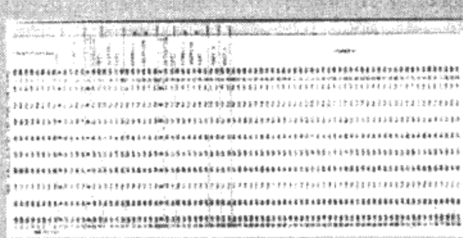
算术逻辑部件

算术逻辑部件（ALU）能执行基本的算术运算，如两个数的加法、减法、乘法和除法。该部件还能执行逻辑运算，如与运算、或运算和非运算。ALU操作的是字，因此计算机的字长是ALU处理的值的大小。Pentium 4的字长是32位或4个字节。

> **算术逻辑部件**（arithmetic/logic unit，ALU）：执行算术运算（加法、减法、乘法和除法）和逻辑运算（两个值的比较）的计算机部件。

Herman Hollerith是谁？

1889年，美国人口调查局认识到，除非他们找到更好的方法进行1890年的人口普查，否则在1900年进行下一次人口普查时，他们将不能用表格列出人口普查的结果。Herman Hollerith基于穿孔卡片设计了一种记数方法。这种方法被用于把人口普查的结果列成表，这种卡片被称为Hollerith卡。Hollerith的电子制表系统导致了当今著名的IBM公司的诞生。要了解更多信息，请访问本书的站点。



图由艾奥瓦大学的Douglas W. Jones提供

大多数现代ALU都有少量的特殊存储单元，称为**寄存器**。寄存器能容纳一个字，用于存放立刻会被再次用到的信息。例如，在计算下列表达式时

One * (Two + Three)

Two首先被加到Three上，然后生成的结果将乘以One。与其把Two和Three相加的结果存储到内存，然后再检索它，与One相乘，不如把结果留在寄存器中，用寄存器的内容乘以One。访问寄存器比访问内存快得多。

> **寄存器**（register）：CPU上的一小块存储区域，用于存储中间值或特殊数据。

输入/输出部件

如果不能把计算中的值从外界输入，或者不能把计算的结果报告给外界，那么任何计算能力都是无用的。输入/输出部件是计算机与外部世界沟通的渠道。

输入部件是使外界数据和程序进入计算机的设备。第一个输入部件所做的是解释纸带或卡片上穿的孔。现代的输入设备包括终端键盘、鼠标和超级市场使用的扫描设备。

输出部件是使外界使用存储在计算机上的结果的设备。最常用的输出设备是打印机和视频显示终端。

> **输入部件**（input unit）：接收要存储在内存中的数据的设备。
>
> **输出部件**（output unit）：一种设备，用于把存储在内存中的数据打印或显示出来，或者把存储在内存或其他设备中的信息制成一个永久副本。

控制器

控制器掌管着读取-执行周期（将在下一节中讨论），因此是计算机中的组织力量。在控制器中有两种寄存器。**指令寄存器**存放的是正在执行的指令，**程序计数器**存放的是下一条要执行的指令的地址。

由于ALU和控制器的协作非常紧密，所以它们常常被看作一种部件，被称为**中央处理器**（Central Processing Unit，CPU）。

图5-2展示了冯·诺伊曼机中各个部分的信息流。这些组成部分由一组电线连接在一起，这组电线被称为**总线**，数据通过总线在计算机中传递。可以把总线看作数据流经的高速公路。尽管数据流动的方式有很多种，但这是一个通用的比喻。

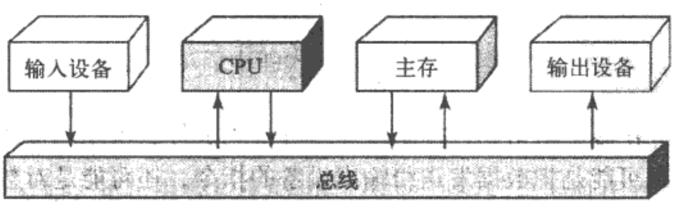


图5-2　冯·诺伊曼机上的数据流

在个人计算机中，冯·诺伊曼机的部件物理驻留在一个印刷电路板上，这个电路板被称为**主板**。此外，主板上还有其他设备（如鼠标、键盘或附加存储设备）与总线的接线。

控制器（control unit）：控制其他部件的动作，从而执行指令序列的计算机部件。

指令寄存器（instruction register，IR）：存放当前执行的指令的寄存器。

程序计数器（program counter，PC）：存放下一条要执行的指令的寄存器。

中央处理器（CPU）：算术逻辑部件和控制器的组合，是计算机用于解释和执行指令的“大脑”。

总线（bus）：把机器的主要组成部分连接在一起的一组电线，数据在这组电线中流动。

主板（motherboard）：个人计算机的主电路板。

5.2.2 读取-执行周期

在仔细研究计算机是如何工作的之前，让我们先看看它能做些什么。计算机的定义略述了它的能力：计算机是一种能够存储、检索和处理数据的设备。因此，给予计算机的指令都与存储、检索和处理数据有关。第7章和第8章将介绍各种用于向计算机发出指令的语言。本章的例子只使用简单的类似英语的指令。

请回忆一下冯·诺伊曼机的原理，即数据和指令都存储在内存中，以同样的方式处理。也就是说，数据和指令都是可以编址的。指令存储在连续的内存区域中，它们操作的数据存储在另一块内存区域中。要启动读取-执行周期，第一条指令的地址将被装入程序计数器。

处理周期中的步骤如下：

- 读取下一条指令
- 译解指令
- 如果需要，获取数据
- 执行指令

让我们更详细地看看每个步骤。整个过程从存储在程序计数器中的第一条指令在内存中的地址开始。

读取下一条指令

程序计数器（PC）存放的是下一条要执行的指令的地址，因此控制器将访问PC中指定的内存地址，复制其中的内容，把副本放入指令寄存器中。此时，指令寄存器存放的是将要执行的指令。在进入周期中的下一步之前，必须更新程序计数器，使它存放当前指令完成时要执行的下一条指令的地址。由于指令连续存储在内存中，所以给程序计数器加1就可以把下一条指令的地址存入PC。因此，控制器将把程序计数器加1。也可能在指令执行完之后才更改PC。

每次内存访问需要花费一个周期。本章开始处的广告中介绍的计算机访问内存的速度是每秒32亿个周期。

译解指令

为了执行指令寄存器中的指令，控制器必须确定它是什么指令。可能是访问来自输入设备的数据的指令，也可能是把数据发送给输出设备的指令，还可能是对数值执行某种运算的指令。在这一阶段，指令将被译解成控制信号。也就是说，CPU中的电路逻辑将决定执行什么操作。这一步解释了为什么一台计算机只能执行用它自己的语言表示的指令。指令本身被逐字地嵌入了电路。

如果需要，获取数据

被执行的指令要完成它的任务，可能需要额外的内存访问。例如，如果一条指令要把某

个内存单元中的内容装入寄存器，控制器就必须得到这个内存单元的内容。

执行指令

一旦译解了指令，并且读取了运算数，控制器就为执行指令做好了准备。执行指令要把信号发送给算术逻辑部件以执行处理。在把一个数加到一个寄存器中的情况下，运算数将被发送给ALU，加到寄存器中的内容上。

当执行完成时，下一个周期开始。如果上一条指令是把一个值加到寄存器中的内容上，那么下一条指令可能是把结果存储在内存中的某处。但是，下一条指令也可能是一条控制指令，询问一个关于上条指令的结果的问题，而且可能会改变程序计数器的内容。

图5-3总结了读取-执行周期。

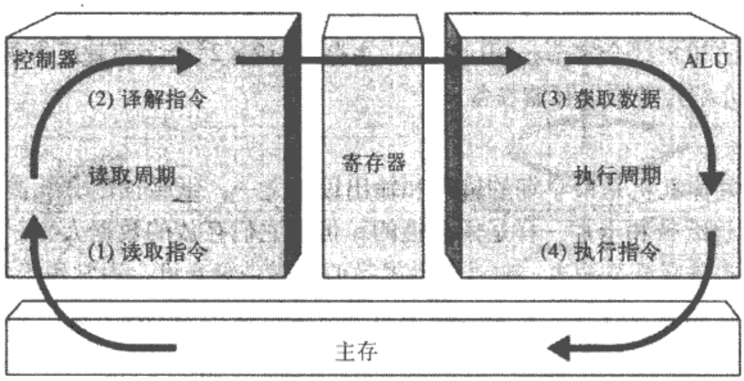


图5-3 读取-执行周期

在过去的半个世纪中，硬件已经发生了翻天覆地的变化，然而冯·诺伊曼机仍然是当今大多数计算机的基础。如著名的计算机科学家Alan Perlis在1981年所说的，“有时，我认为计算领域内的唯一通则就是读取-执行周期”。[3]即使在20多年后的今天，这句话仍然是正确的。

5.2.3 RAM和ROM

前面介绍过，RAM是随机存取存储器的缩写，这是一种每个存储单元能够被直接访问的内存。访问每个存储单元的本质，是改写这个存储单元的内容。也就是说，把其他数据存入这个单元，改变其中的位组合。

请回忆一下，数据和指令都驻留在内存中，以同样的方式处理。这意味着在程序执行过程中，指令可能改变。怎么会出现这种情况呢？可能存在这样的指令，要读取的存储单元存放的是另一条指令，当前指令对这个单元中的内容进行加或减操作后，把结果又存回这个单元。有时，你可能真的想这样做。但是，随意地更改程序代价很高。ROM内存则可以解决这个问题。

ROM是只读存储器（Read Only Memory）的缩写。ROM中的内容不能更改，是永久的，存储操作不能改变它们。把位组合放在ROM中称为“烧入”。只有在制造ROM或装配计算机时才能烧入位组合。

还有一个非常基本的属性可以用来区分RAM和ROM。RAM具有易失性，而ROM则没有。也就是说，关闭电源后，RAM不再保留它的位配置，但是ROM仍然保留这些配置。ROM中的位组合是永久性的。由于ROM稳定，不能更改，所以用它存储计算机启动自身需要的指令。经常使用的软件也存储在ROM中，以免每次开机都要读取软件。

主存通常包含一些ROM和通用的RAM。注意，ROM也是随机访问的。

5.2.4　二级存储设备

如前所述，输入设备是数据和程序进入计算机并存储在内存中的途径。输出设备则是把结果发送给用户的途径。由于大部分主存都是易失的、有限的，所以还需要其他类型的存储设备，当不再处理程序和数据或关机时，把程序和数据保存起来。这些类型的存储设备（除了主存）称为*二级存储设备*或*辅助存储设备*。由于必须从这些存储设备中读取数据，并把数据写回，所以每个二级存储设备也是一种输入和输出设备。

二级存储设备可以在工厂时就安装到机箱中，也可以需要时再添加。因为这些存储设备可以存放大量的数据，所以它们又被称为大容量存储设备。例如，广告中的计算机附带的硬盘驱动器能够存储80×2^{30}个字节，相比之下，主存的512×2^{20}个字节就显得很少了。

接下来我们介绍几种二级存储设备。

磁带

读卡器和卡片穿孔机是最早期的输入和输出设备之一。纸带读出穿孔器是下一代的输入和输出设备。尽管纸带和卡片一样是永久性的，但是它们存放的数据太少。第一种真正的大容量辅助存储设备是*磁带驱动器*。磁带驱动器类似于磁带录音机，通常用于备份（生成副本）磁盘上的数据，以防磁盘损毁。磁带的类型多种多样，从小型的流式录音带，到大型的盘式磁带。

磁带驱动器有一个严重的缺点，即如果要访问磁带中间的数据，则必须访问这个数据之前的所有数据并丢弃它们。虽然现代的流式磁带系统能够跳读磁带片段，但从物理上讲磁带仍然要经过读写头。磁带的任何物理移动都是费时的，如图5-4所示。

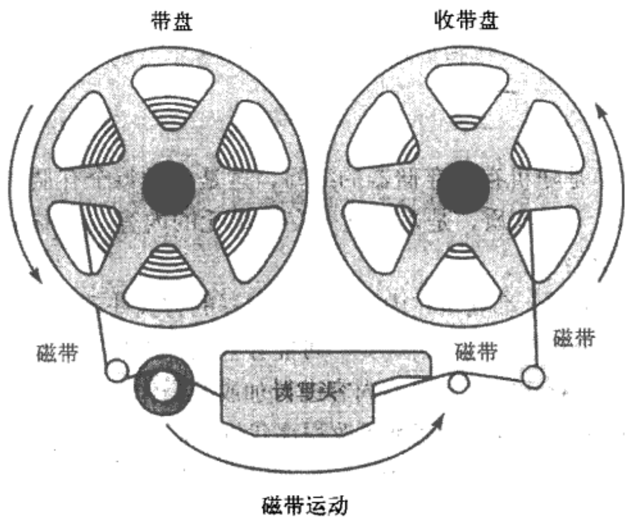


图5-4　磁带

磁盘

*磁盘驱动器*是CD播放器和磁带录音机的混合物。读写头（相当于磁带录音机中的录音/回放头）通过在高速旋转的磁盘上移动来检索和记录数据。与CD一样，读写头能直接访问想得

到的信息，此外，还与磁带一样，信息是被磁化存储的。

尽管磁盘种类不一，但是它们使用的都是由磁质材料制成的薄磁盘。每个磁盘的表面都被逻辑划分为**磁道**和**扇区**。磁道是磁盘表面的同心圆。每个磁道被分为几个扇区。每个扇区存放一个信息**块**，是连续的位序列（如图5-5a所示）。虽然越靠近圆心的磁道看起来越小，但是每个磁道中的扇区数是相同的，每个扇区中的位数也是相同的。越靠近圆心的数据块数据排放得越密集。每个磁盘表面的磁道数和每个磁道中的扇区数可能不同，通常使用的是512字节或1024字节（同样是2的幂）。在格式化磁盘时，将用磁性标示磁道和扇区中的区域，从物理上来说，它们不属于磁盘。

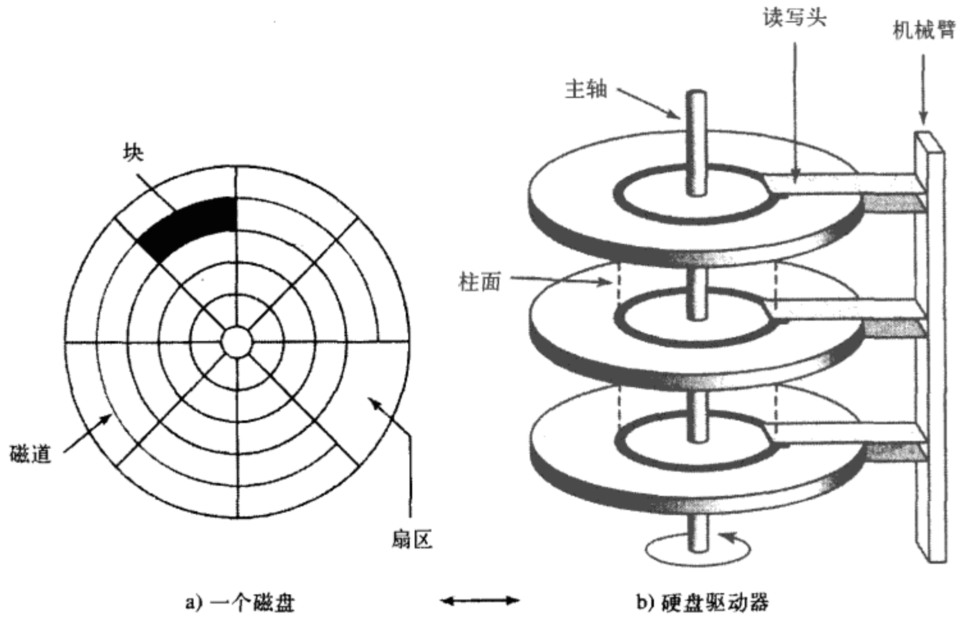


图5-5 磁盘的结构

> **磁道**（track）：磁盘表面的同心圆。
> **扇区**（sector）：磁道的一个区。
> **块**（block）：存储在扇区中的信息。

磁盘驱动器中的读写头固定在一个机械臂上，机械臂可以从一个磁道移动到另一个磁道（如图5-5b所示）。输入或输出指令将指定磁道和扇区。当读写头经过正确的磁道时，将等待正确的扇区转动到读写头下，然后访问该扇区中的信息块。这一过程产生了四种衡量磁盘驱动器效率的方法：**寻道时间**、**等待时间**、**存取时间**和**传送速率**。寻道时间是读写头定位到指定的磁道所花费的时间。等待时间是读写头等待指定的扇区转到其下所花费的时间。平均等待时间是磁盘旋转一圈需要的时间的一半。因此，等待时间又称为*旋转延迟*。存取时间是寻道时间和等待时间之和。传送速率是把数据从磁盘传输到内存的速率。

> **寻道时间**（seek time）：读写头定位到指定的磁道所花费的时间。
> **等待时间**（latency）：把指定的扇区定位到读写头之下所花费的时间。

存取时间（access time）：开始读取一个数据块之前花费的时间，即寻址时间和等待时间之和。
传送速率（transfer rate）：数据从磁盘传输到内存的速率。
柱面（cylinder）：所有磁盘表面的同心磁道的集合。

现在，让我们来看看各种磁盘。磁盘的分类之一是硬盘和软盘。这些术语指磁盘本身的柔韧性。20世纪70年代引入了最初的软盘，直径8"，连外壳都是软的。70年代末个人计算机出现后，软盘的直径减小成了5.5"。目前的软盘直径为3.5"，封装在硬塑料壳中，能够存储1.44MB数据。与几年前不同的是，较新的机器不再具有内置的软盘驱动器，不过软盘仍在流行，可以给机器附加软盘驱动器。当今还有一些较新的专用磁盘，例如Zip盘和它的驱动器。Zip盘可以在一个磁盘上存储几百MB的数据，不过Zip盘很贵。

计算机安装的硬盘由几个磁盘构成，听起来有些奇怪，我们来解释一下。单个的磁盘被称为磁盘片。硬盘由几个连接在旋转主轴上的磁盘片构成。每个磁盘片有自己的读写头。上下排列的所有磁道形成了一个**柱面**，如图5-5b所示。硬盘上的地址由柱面编号、表面编号和扇区构成。硬盘驱动器的旋转速度比软盘驱动器快得多，读写头并不真的接触磁盘片的表面，而是在上面飘浮过。常见的硬盘驱动器转速是每分钟7200转，软盘驱动器的平均转速只是它的十分之一。因此，在提到硬盘传送速率时，通常使用每秒多少兆字节，而在提到软盘传送速率时，使用的则是每秒多少千字节。

CD和DVD

可以用浓缩汤来比喻光盘和它们的驱动器。前面分析的广告中使用了两个缩写：CD-ROM和CD-RW。此外，还需要解释CD-DA、CD-WORM和DVD。当你读完这本书后，会知道更多的缩写。

让我们先看看缩写CD。CD当然是光盘（Compact Disk）的缩写，你可能收藏了大量的音乐CD。CD驱动器使用激光读取存储在塑料盘片上的信息。CD上面没有同心磁道，而只有一个从里向外盘旋的螺旋磁道。与其他盘片一样，这个磁道被划分为扇区。与磁盘不同的是，盘片中心附近的磁道中的数据并不密集，CD中的数据是均匀分布在整个光盘上的，因此磁道外边缘处存储的信息比较多，一转读到的信息也比较多。为了使整个光盘的传送速率一致，盘片的旋转速度会根据光束的位置而变化。

附加在CD后的其他字母说明了光盘的各种属性，如格式或其上的信息是否可以更改等。CD-DA是数字音频光盘（Compact Disc Digital Audio）的缩写，说明了录音采用的格式。这种格式中的某些域用于时间安排。CD-DA中的一个扇区可以存放一秒音乐的1/75。

CD-ROM与CD-DA一样，只是格式不同。在CD-DA中，存储在扇区中的数据是为时间安排信息预留的。ROM是只读存储器的缩写。在介绍广告中的CD-ROM时提到过，只读存储器中的数据是永久存储在光盘上的，不能改变。CD-ROM上的一个扇区能存放2KB数据。广告中给出了CD-ROM的传送速率，但是没有给出驱动器读取的光盘的容量。不过，它的容量大约为600MB。

DVD是现在最常用的分发电影的格式，最多可以存放10GB的数据。DVD是数字化视频光盘（Digital Versatile Disc）的缩写，能够存放集合了音频和视频的多媒体信息。

CD-WORM是一次写入多次读出光盘（Write Once，Read Many）的缩写。使用这种技术，可以在制造出CD之后才刻录信息。这种格式通常用于归档数据，数据刻录之后不能再更改。

最后，缩写RW或RAM表示光盘是既能读又能写的。

注意，CD-ROM和DVD-ROM的速度单位是X，例如前面示例说明中所用的，它表示标准的音频CD和DVD播放器的速度。在评估这类设备时，列出的速度是一个最大值，表示读取光盘上的某些部分的速度。它们并非平均值。因此，在衡量性价比时，读盘速度越快并不表示越好。

笨人的笔记本电脑

笔记本电脑用户最常报告的故障是摔坏了（占报告故障的用户的60%以上），其次是撒上了液体（占报告故障的用户的50%）。笔记本电脑制造商现在出售“半粗糙”的笔记本，这种机器防摔防泼洒。这种机器比普通的笔记本电脑重一些，也更贵一些，这种新机器的特点有：

- 它在自动探测到下跌时，会立刻保护硬盘（类似于汽车中的安全气囊）。
- 防水键盘。
- 机器内部有一个硬的镁制框架，作用就像一个滚柱罩。

5.2.5 触摸屏

我们已经知道了二级存储设备如何为CPU使用的数据和程序提供存储单元。使用其他输入输出（I/O）设备则允许用户与正在执行的程序进行交互。有许多常见的例子，如通过键盘和鼠标提供信息，通常阅读显示器显示的信息。其他的输入设备包括条码阅读器和图像扫描仪，输出设备则包括打印机和绘图器。

我们要详细介绍的是一种特殊的I/O设备——*触摸屏*。它显示文本和图形的方式与常规的显示器相同，此外，它还能探测到用户在屏幕上用手指或书写笔的触摸，并做出响应。通常，一个I/O设备只能担任输入设备或者输出设备，但是触摸屏则兼具两者的功能。

你可能在各种情况下见过触摸屏，如在信息亭、餐馆和博物馆。图5-6展示了一位用户正在使用触摸屏。在需要复杂的输入的情况下，触摸屏非常有用，它还有一个好处，就是被保护得相当好。餐厅中的服务生如果用触摸屏点菜，比用键盘要好得多，键盘上的按键远远多于完成点菜这样的任务所必需的数量，而且食物和饮料很容易使键盘损毁。

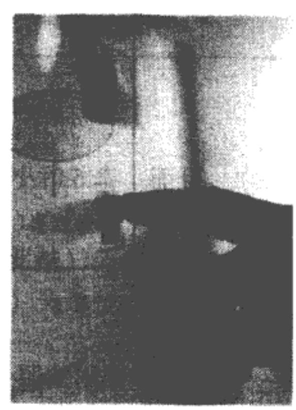


图5-6 触摸屏

触摸屏并非只能检测到触摸，它还能知道触摸屏幕的位置。通常用图形化的按钮来表示选项，让用户通过触摸屏幕上的按钮做出选择。在这个方面，触摸屏与鼠标没什么区别。跟踪鼠标的移动可以得到鼠标的位置，当点击了鼠标按钮时，鼠标指针的位置将决定按下的是哪一个图形化按钮。在触摸屏上，触摸屏幕的位置决定了按下的按钮。

那么，触摸屏是如何检测到触摸的呢？此外，它如何知道触摸屏幕的位置呢？目前用来实现触摸屏的技术有几种，我们来简短地探讨一下这些技术。

*电阻式*触摸屏由两个分层构成，每个分层由导电材料制成，一层是水平线，一层是竖直线。两个分层之间有非常小的空隙。当上面的分层被按下后，它将与下面的分层接触，使电流流通，接触的竖直线和水平线说明了触摸屏幕的位置。

电容式触摸屏在玻璃屏幕之上附加了一个层压板，它可以把电流导向任何方向，而且屏幕的四角还有等量的微弱电流。当屏幕被触摸时，电流将流向手指或书写笔。电流流动得非常缓慢，用户甚至感觉不到这种电流。触摸屏幕的位置是靠比较来自每个角的电流的强弱确定的。

红外触摸屏把十字交叉的水平和竖直红外光束投射到屏幕的表面。屏幕反面的传感器将探测光束。用户触摸屏幕时，会打断光束，此时能够确定断点的位置。

声表面波（SAW）触摸屏与红外触摸屏相似，只不过它投射的是在水平和垂直坐标轴上相交的高频声波。当手指触摸到屏幕时，相应的传感器将检测到断点，并确定触摸的位置。

注意，可以用带手套的手指触摸电阻式触摸屏、红外触摸屏和声表面波触摸屏，但不能用到电容式触摸屏上，因为它依靠的是流向触摸点的电流。电阻式触摸屏和电容式触摸屏的表面都是特殊材料，容易受损，而红外触摸屏和SAW触摸屏的表面都是标准的玻璃，因此不容易受物理损毁。

5.3 非冯·诺伊曼体系结构

现今，冯·诺伊曼体系结构的线性读取-执行周期在技术领域仍然占统治地位。但是，从1990年起，并行处理系统进入了市场。它们有能力以更高的速度处理更多的数据。

一种并行方法是用多个处理器把同一个程序应用到多个数据集上。采用这种方法，处理器通常同时执行相同的指令，也就是说，一个公共程序在每个处理器上运行。这种方法被称为**同步处理**，当同一个进程需要应用到多个数据集时，这种方法有效。请参阅图5-7。这种方法与NASA的后援系统相似，这种系统使用三台计算机做同样的事情，以此作为安全机制。不过，这里是用多个处理器把同样的进程并行应用到不同的数据集。

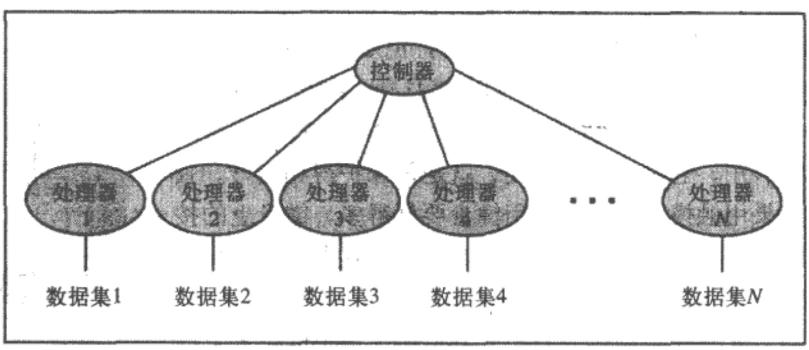


图5-7　同步处理环境中的处理器

另一种配置则按照一前一后的方法安排处理器，使每个处理器负责整个计算过程中的一部分。这种方法称为**流水线操作**，令人回想起工厂的装配线。用这种方法处理数据时，第一个处理器将负责第一个任务。第二个处理器将以第一个处理器输出的结果开始工作，而第一个处理器则开始对下一个数据集进行运算。最后，每个处理器负责作业的一个阶段，从前一阶段的处理获取素材或数据，然后顺次把自己的工作转入下一个阶段。如图5-8所示。

同步处理（synchronous processing）：用多个处理器把同一个程序同时应用到多个数据集。

流水线操作（pipelining processing）：一前一后地安排多个处理器，使每个处理器负责整个运算的一部分。

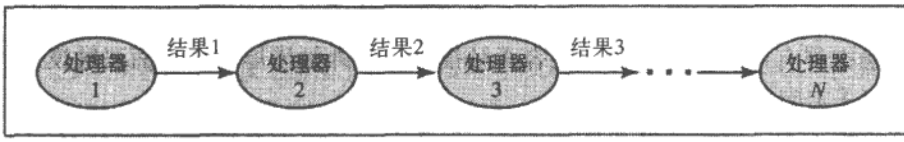


图5-8 流水线操作中的处理器

在同步处理的例子中，每个处理器对不同的数据集做同样的操作。例如，每个处理器可能计算不同班级的分数。在第二个例子中，每个处理器计算的是同一个班的分数。第三种方法是让不同的处理器对不同的数据进行不同的操作。使用这种配置，处理器大部分时间可以独立工作，但会出现处理器之间的协调问题。这使得每个处理器有一个本地内存和一个共享内存。处理器通过共享内存进行通信，因此，这种配置称为**共享内存**配置。如图5-9所示。

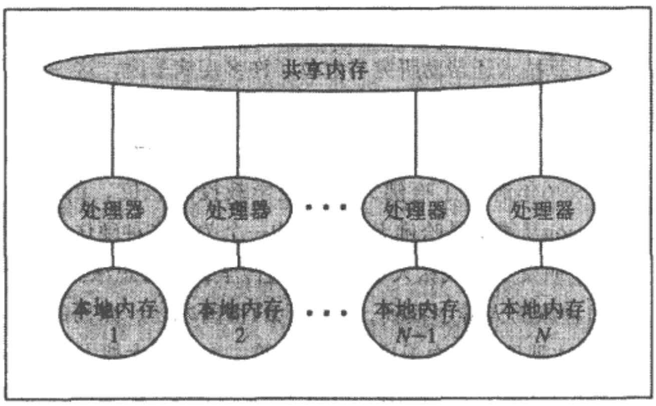


图5-9 处理器的共享内存配置

> **共享内存**（shared memory）：多个处理器共享一个全局内存。

小结

构成计算机的部件涉及多种设备，它们各有特征，包括速度、大小和效率。而且，它们在计算机的整体操作中各自扮演着必不可少的角色。

计算的世界中充斥着各种行话和缩写。处理器的速度以GHz（千兆赫）为单位，内存的容量以MB（兆字节）为单位，外部存储设备的容量以GB（千兆字节）为单位，显示器的质量则以dp（像点间距）衡量。

冯·诺伊曼体系结构是当今大多数计算机的底层体系结构。它有5个主要组成部分：内存、算术逻辑部件、输入设备、输出设备和控制器。在控制器指挥下的读取-执行周期是这个处理过程的核心。在这个周期中，将从内存读取指令，译解指令，执行指令。

RAM和ROM是两种计算机内存的缩写。RAM表示随机存取存储器；ROM表示只读存储器。存储在RAM中的值是可更改的；存储在ROM中的值则不可更改。

二级存储设备对计算机来说至关重要。这些设备在计算机不运行的时候保存数据。磁带和磁盘是两种常用的二级存储介质。

触摸屏是一种外围设备，同时具备输入和输出功能，适用于餐厅和信息亭这种特定环境。

它们能对手指或书写笔对屏幕的触摸做出响应，并且能确定触摸屏幕的位置。现有的触摸屏技术实现的触摸屏有电阻式触摸屏、电容式触摸屏、红外触摸屏和声表面波触摸屏。它们各有特点，适用于不同的情况。

虽然目前为止冯·诺伊曼机是最常见的，但还存在其他体系结构。例如，有些机器的处理器不止一个，所以能够进行并行计算，从而加速处理过程。

道德问题：生物信息学研究和deCODE Genetics公司的案例

由于计算机/信息技术和生物技术出现了交点，所以出现了综合两者的技术。其中一个领域就是生物信息学，它涉及用计算机和计算机网络获取、存储、处理、分析、虚拟化和共享生物信息。许多生物研究（包括遗传和染色体在内）现在都是通过计算技术和建模技术（也就是使用生物物质的数字表示法）来执行的，而不是采取用化学方式执行的传统的“湿”实验室技术。到2003年为止，染色体研究员已经可以用计算机工具绘制完整的人体染色体组。如果使用传统的排序方法，还需要更多年来实现这一目标。计算技术还帮助研究员定位了许多疾病基因，这一成果使治疗和治愈这些疾病的制药业飞速发展。不过，在遗传和染色体研究中采用计算机技术还存在一定的争议，例如deCODE Genetics公司这个案例。

Kari Stefansson于1996年辞去了哈佛大学神经学教授的职位，创立了私营的遗传学公司deCODE Genetics。Stefansson从事的是哈佛大学资助的一个关于多种硬化症的研究，由于能够访问的人群的数据有限，致使他萌发了寻找私人资助创建遗传学公司的想法。Stefansson相信，适合基因学研究的资源可以在同种族的人群中找到，例如他的祖国冰岛。冰岛的280 000名居民的祖先（Norse和Celts）是9世纪到达那里的。由于这个国家在第二次世界大战之前一直处于一种与世隔绝的状态，所以极大地保留了同类性，从而为像Stefansson这样的染色体研究者提供了理想的资源。

Stefansson与冰岛政府进行了商谈，要求访问1915年之后这个国家的健康护理档案。他已经访问了冰岛的家系数据，其中包括可以追溯到1000多年前的记录。冰岛的国会还承诺Stefansson可以访问任何政府所有的有助于研究的医疗信息。有了这些协议，Stefansson成立了deCODE Genetics公司，立即开始构建一个基因数据库，其中将包括最终从70 000名志愿者采集到的DNA样品中获取的信息。然后deCODE公司就能把存储在三个独立数据库中的医疗/健康护理档案、家系档案和遗传信息联系在一起，建立交叉引用。

那么，为什么从道德和社会学的角度看，deCODE公司的案例会存在争议呢？这个案例引起的道德和社会学问题可以从三个不同但又相关的观点来看。第一，它冒犯了自愿提供DNA的研究对象的隐私权和机密性。第二，它违反了采集个人遗传数据所采用的认可策略和方法。第三，它侵犯了存储在计算机数据库中的个人遗传信息的所有权。

首先，来看看关于隐私权的问题。自愿为deCODE提供DNA样本的研究对象都有一个假设，即这些个人遗传数据是机密信息，会得到相应的处理和保护。但是，这些人并没有得到确保他们的个人数据保持机密的保证，也没有得到在通过数据挖掘技术对这些数据与deCODE公司的其他非遗传数据的数据库进行交叉引用时也会继续保护它们的保证（集合信息并不享受与“个人身份信息”同等的法律保护）。集合信息会生成关于人的“新事实”，从而根据武断的、非显见的模式和统计相关性产生“新分组”。例如，数据挖掘可能会揭示一个统计模式，即阅读19世纪的英国小说的妇女多发乳腺癌。结果就是这类人会被划分到一个新的组或分类，即阅读19世纪的英国小说和多发乳腺癌的妇女这种分组。很可能这些妇女并不知道存在这样的分组，而一些保险公司及其雇员会知道这种武断的统计相关性。由于这些妇女与这个分组的相关性，她们没有得到适当的隐私保护，在就业和

申请健康保险时就存在被拒绝的危险。

其次，至少在两种不同程度上违反采集数据所采用的认可方法。其一，deCODE从冰岛政府获取的健康护理数据库中的医疗信息是以假定（而不是通知）认可所获取的数据为基础的。其二，从“通知的”志愿者那里获取的与DNA有关的信息并不满足医疗伦理学家描述的“有效通知认可”的条件。即使那些把自己的遗传数据贡献给deCODE的研究对象同意在特定的环境中使用他们的数据，但他们也没有明确认可过可以在第二种或接下来的各种环境中（可能是由数据挖掘造成的）使用这些数据。也就是说，他们并没有授权允许这些个人数据与其他个人医疗数据（如健康护理和家系数据库中的电子记录）交叉引用。他们没有授权对交叉引用的信息进行“挖掘”。从这点来看，deCODE公司采用的认可方法对研究对象来说是不透明的。

再次，关于个人遗传数据所有权的问题。例如，谁可以（谁不可以）访问这些数据？在deCODE案例中，个人遗传数据存在于公司数据库中。那么这家公司对存储在自己数据库中的个人遗传信息具有唯一所有权吗？如果是这样，那么这家公司是否具有永久所有权？是否应该允许deCODE用那些得到的数据从事任何想做的事呢？此外，是否应该授予冰岛政府对保存在deCODE数据库中的个人遗传信息的保管权，以便更好地保护冰岛公民的利益？至于自愿提供DNA样本的人，根本不清楚他们是否对如何使用自己的个人遗传数据具有任何控制权。

deCODE公司的案例呈现出了生物信息研究在个人范畴内的一些重要问题和观点。例如，是否应该修改或扩展隐私权法案来保护研究对象？由于计算技术可以用于再次利用个人遗传数据，那么是否需要制定新的隐私权法案呢？是否需要更新和修改通知认可的政策呢？是否需要为私营企业建立审查委员会来保护个人，就像在学术领域为政府资助的研究项目设立的制度审查委员会（IRB）一样？根据个人遗传数据可能保存在商业数据库中这一事实，是否需要修改所有权法案？也许现在考虑这些还为时过早，毕竟生物信息学还是一门新兴学科。

这个案例采用的资料摘自H. Tavani编写的《Ethics,Computing, and Genomics》(Sudbury, MA: Jones and Bartlett, 2006)。

练习

为练习1～16中的名称或用途找到匹配的10的幂。

A. 10^{-12}　　B. 10^{-9}

C. 10^{-6}　　D. 10^{-3}

E. 10^{3}　　F. 10^{6}

G. 10^{9}　　H. 10^{12}

I. 10^{15}

1. Nano
2. Pico
3. Micro
4. Milli
5. Tera
6. Giga
7. Kilo
8. Mega
9. 常用于描述处理器速度。
10. 常用于描述内存大小。
11. 用于说明Internet速度。
12. 拉丁语中的一千。
13. 西班牙语中的很少的。
14. Peta。
15. 约等于2^{10}。
16. 约等于2^{50}。

为练习17～23中的定义找到与之匹配的缩写。

A. CD-ROM　　B. CD-DA

C. CD-WORM　　D. DVD

E. CD-RW　　F. CD

17. 在制造过程中刻录的一般光盘。
18. 存储在扇区中的数据是预留的时间安排信息。
19. 可以读多次，但只能在出厂后写一次。
20. 可以进行多次读写操作。

21. 录音使用的格式。
22. 只有一个从内到外盘旋的磁道。
23. 可以存储大量多媒体数据的新技术。

练习24～64是问题或简答题。

24. 定义下列术语：
    a) Pentium 4处理器
    b) 赫兹
    c) 随机存取存储器
25. Pentium 4处理器中的字长是多少？
26. 处理器是1.4GHz表示什么意思？
27. 内存是133MHz表示什么意思？
28. 下列机器的内存是多少字节的？
    a) 128MB机
    b) 256MB机
29. 定义RPM，并讨论在访问磁盘的速度方面，它表示什么意思。
30. 存储程序的概念是什么？为什么它很重要？
31. 在计算机体系结构中，“处理信息的部件从存储信息的部件中分离出来了”，请解释这句话的意思。
32. 列出冯·诺伊曼机中的部件。
33. 8位机的可编址性是多少？
34. ALU的功能是什么？
35. 在冯·诺伊曼机中担任舞台总监角色的部件是什么？请解释它的功能。
36. 穿孔卡片和纸带是早期的输入或输出介质。请讨论它们的优点和缺点。
37. 什么是指令寄存器，它的功能是什么？
38. 什么是程序计数器，它的功能是什么？
39. 列出读取－执行周期中的步骤。
40. 请解释什么是“读取一条指令”。
41. 请解释什么是“译解一条指令”。
42. 请解释什么是“执行一条指令”。
43. 请比较RAM和ROM的异同。
44. 什么是二级存储设备？为什么这种设备很重要？
45. 请讨论用磁带作为存储介质的优点和缺点。
46. 请绘制一个磁盘的表面，标出磁道和扇区。
47. 请定义磁盘上的数据“块”。
48. 什么是柱面？
49. 请描述把一个数据块从硬盘传递到内存的步骤。
50. 请区分光盘和磁盘。
51. 请描述采用同步处理的并行体系结构。
52. 请描述采用流水线操作的并行体系结构。
53. 共享内存配置是如何运作的？
54. 一个16位的处理器能够访问多少个内存区域？
55. 在对计算机广告的讨论中，我们使用了三次“越快越好”。请分别解释在每种情况下它的含义。
56. 在对计算机广告的讨论中，我们用“越小越好”来形容与显示器相关的参数，请解释它的含义。
57. 在对计算机广告的讨论中，我们用“越大越好”来形容与光盘相关的参数，请解释它的含义。
58. 请记录术语硬件和软件一周之中在电视广告中出现的次数。
59. 请找一份当前台式电脑的广告，与本章开头处的广告进行比较。
60. 作为二级存储设备的磁盘的通用名是什么？
61. 术语像点间距指的是什么？
62. 什么是调制解调器？
63. 是下载速度快，还是上载速度快？
64. 1KB内存和1KB传送速率有什么区别？

思考题

1. 在表示一个16位处理器中的地址时，是采用八进制好，还是采用十六进制好，请解释你的答案。
2. 请把程序的概念和冯·诺伊曼机的读取－执行周期联系在一起。
3. 最初的个人计算机装配有一个软盘驱动器，之后装配了两个软盘驱动器。此后，当CD驱动器加入标准配置后，软盘驱动器成了可选的配件。你认为未来个人计算机的标准配置是什么？
4. 为什么在引用存储容量时只使用10的幂？难道10的幂和2的幂不够近似吗？
5. 对于制药公司的电视广告，你有何看法？
6. 你同意参与deCODE的研究吗？
7. 个人可以通过什么方法保护自己的医疗记录隐私？

第四部分 程序设计层

第6章 问题求解和算法设计

第6章是介绍程序设计层的第一章。第2章和第3章介绍了理解计算系统必需的信息，包括记数系统和在计算机中如何表示各种类型的信息。第4章和第5章介绍了计算机的硬件部件。接下来，重点将从计算机系统是什么转移到如何使用它。

这一章将分析问题求解的方法以及如何用*伪代码*编写解决方案（算法）。伪代码看起来就像菜谱。菜谱是一套指令，说明如何烹饪一道比如以鱼为主的菜。与菜谱相似，伪代码也是一套指令，说明如何解决一个特定的问题。在这一个分层接下来的几章中，将介绍如何把伪代码翻译成一种计算机能够运行的程序设计语言。

目标

学完本章之后，你应该能够：

- 确定一个问题是否适合用计算机解决。
- 结合Polya提出的如何解决问题的列表，描述计算机问题求解的步骤。
- 区别执行算法和开发算法。
- 描述用于表示算法的伪代码指令。
- 使用伪代码表示算法。
- 应用自顶向下的方法开发算法来解决问题。
- 定义面向对象设计方法中的术语。
- 应用面向对象的方法开发一组互动对象来解决问题。
- 求证与问题求解相关的几点思想——信息隐蔽、抽象、事物命名和测试。

6.1 问题求解

对你来说，*问题求解*是什么意思呢？是不是唤起了你的回忆，想到一个孩子无精打采地做着数学作业？还是想到一个农夫努力地要在暴风雨到来之前收起自己收获的干草？想到妈妈面对涨了的学费在制定新的预算？想到爱因斯坦在绞尽脑汁地研究着相对论？想到自己要把邀请参加游戏的7个人塞到自己的四座车内？想到自己的男朋友或女朋友看来要和你分手了？想到你答应明天给俱乐部的地址列表？想到苏丹达尔富尔地区的种族灭绝？抑或想到伊拉克和巴勒斯坦间的冲突？

问题在字典中的定义是为调查、思考和解决而提出的难题。在数学中，问题通常是用明确的数学法则来解决的情况。还有一种定义，说问题是复杂的、未解决的难题，或令人感到困惑、痛苦及烦恼的来源。综合这些定义，**问题求解**就是找到令人感到困惑、痛苦、烦恼或

未解决的难题的解决方案的行动。

> **问题求解**（problem solving）：找到令人感到困惑的难题的解决方案的行动。

当然，第一段中的所有难题都符合这个定义。但是，它们中只有一部分适用于计算环境。计算机不能用于（至少不能直接用于）帮助农夫收干草，帮助把多余的人塞进车子，帮助将要分手的男女朋友，帮助平息苏丹的战事，帮助解决伊拉克和巴勒斯坦之间的宗教和领土冲突。涉及物理行为和情感的难题，计算机都不能解决。

此外，如果不告诉计算机要做什么，它什么也做不了。计算机是没有智能的，它不能分析问题并产生问题的解决方案。人（程序员）必须分析这些问题，为解决问题开发指令集（程序），然后让计算机执行这些指令。

如果计算机不能解决问题，那么计算机还有什么用呢？其实，只要为计算机编写了解决方案，它就能够对不同的情况和数据快速一致地反复执行这个方案，它把人们从枯燥重复的任务中解放了出来。

让学生与计算机显示的数学作业进行互动，还能增加数学作业的趣味性。学生看到屏幕上的作业后，可以输入解决方案。程序将检查答案，给出反馈信息。计算机还可以记录日常开销，接管了这个费时的任务后，它就能帮助妈妈制定开支预算了。有大量的商业软件包可以提供这种功能。

如果爱因斯坦能够使用当今任何一台具有增强数学功能的超级计算机，谁能预测这可以节省多少时间呢？对于你为俱乐部准备的地址列表又如何呢？有许多软件包可以用来组织地址列表这样的信息。此外，编写一个处理地址列表的小程序是个非常简单的任务。计算机系的任何一个二年级学生都可以在一周之内编写出一个这种程序的简单版本。

6.1.1 如何解决问题

1945年，G. Polya写了一本书，名为《How to Solve It: A New Aspect of Mathematical Method》（如何解决它：数学方法的新视点）。[1]尽管这本书写于50多年前，当时计算机还处于试验阶段，但是其中关于问题求解过程的描述则非常经典。图6-1总结了这一过程。

Polya的书的经典之处在于，他的“如何解决它”这个列表是普遍适用的。虽然是在解决数学问题这个背景下编写的，但是如果把文字“未知量”换成“问题”，把“数据”换成“信息”，把“定理”换成“解决方案”，那么这个列表就完全适用于各种类型的问题。当然，其中的第二步（找到信息和解决方案之间的联系）是问题求解的核心。让我们仔细看看Polya列表建议的几点策略。

提出问题

如果口头地给你一个问题或任务，通常你会提问，直到自己完全明白了要做什么为止。在任务完全明确之前，你会问何时、为什么、在哪里之类的问题。如果给你的指令是书面的，那么你可能会在空白的地方加个问号，用下划线标出一个单词、一组单词或一个句子，或者用其他方法标示出任务中不明确的地方。也许后面的段落会对你的问题给出答案，也许你必须和提出任务的人进行讨论。如果任务是你自己设置的，那么提出问题的方式不会是口头的，而是下意识的。

下面是一些典型的问题：

- 对这个问题我了解多少?
- 要找到解决方案我必须处理哪些信息?
- 解决方案是什么样的?
- 存在什么特例?
- 我如何知道已经找到解决方案了?

如何解决它

第一步 必须理解问题。	理解问题 未知量是什么？数据是什么？条件是什么？条件有可能满足吗？条件足够决定未知量吗？抑或条件不够决定未知量吗？抑或条件是多余的？抑或条件是与未知量矛盾的？绘制一幅图，引入合适的符号，把条件分割成多个部分。能把它们写下来吗？
第二步 找到信息和解决方案之间的联系。如果找不到直接的联系，则可能需要考虑辅助问题。最终，应该得到解决方案。	设计方案 以前见过这个问题吗？或者以前见过形式稍有不同的同样问题吗？ 知道相关的问题吗？知道可能解决这个问题的定理吗？仔细研究未知量，试想一个所熟悉的、具有同样未知量或类似未知量的问题。 有一个曾经解决过的相关问题。可以使用它吗？可以使用它的结果吗？可以使用它的方法吗？为了使用它，需要引入辅助元素吗？能重述问题吗？能换一个方式叙述问题吗？回到定义。如果不能解决这个问题，先尝试解决一些相关的问题。可以想象一个比较容易解决的相关问题吗？一个更普适的问题？一个更专用的问题？一个相似的问题？可以解决部分问题吗？只保留一部分条件，舍弃其他条件；未知量又明确了多少；它是如何变化的？能从数据得到一些有用信息吗？可以想出另外一些能够确定未知量的数据吗？可以改变未知量或数据，或者同时改变两者，使新数据和新的未知量更接近吗？是否使用了所有数据？是否使用了所有条件？是否考虑到了该问题涉及的所有关键概念？
第三步 执行方案。	执行方案 执行解决方案，检查每个步骤。可以清楚地看到每个步骤都正确吗？可以证明它是正确的吗？
第四步 分析得到的解决方案。	回顾 能检查结果吗？能检查参数吗？可以得到不同的结果吗？看到过这些结果吗？可以用这个结果和方法解决其他的问题吗？

图6-1 Polya的“如何解决它”列表

George Polya



George Polya于1887年12月13日出生在布达佩斯。虽然他是闻名于世的数学家，但他在年少时并没有很早地显示出对数学的兴趣。他的三位中学数学老师给他留下的印象是“两个很拙劣，一个还好”，这一点大概可以解释为什么他对数学缺乏兴趣。

1905年，Polya进入了布达佩斯大学，在妈妈的坚持下，开始研习法律。经过一个枯燥乏味的学期后，他决定改学语言和文学。虽然他具有拉丁语和匈牙利语的教学执照，但却从来没用过。之后，他对哲学产生了兴趣。作为哲学研究的一部分，他上了数学课和物理课。最后，他选择了数学，用他自己的话说“我太擅长哲学，但不擅长物理，数学则

居于两者之间”。1912年，他获得了数学博士学位，从而开始了自己的职业生涯。

Polya曾经在哥廷根大学、巴黎大学和苏黎世的瑞士技术联合会（Swiss Federation of Technology）执教并进行研究工作。在苏黎世时，他教过John von Neumann，他对von Neumann的评价是“Johnny是唯一一个曾经令我感到害怕的学生。如果我在课堂上陈述了一个未解决的问题，那么很可能一下课他就会来找我，交给我一些小纸片，上面潦草地写着那个问题的完整解决方案。”

像许多欧洲人一样，德国的政治环境迫使他于1940年移居到了美国。在Brown大学教授了两年的课程后，他移居到Palo Alto，开始在斯坦福授课，并在此度过了余下的职业生涯。

Polya的研究和出版物涉及数学中的多个领域，包括数论、组合数学、天文学、概率论、积分函数和偏微分方程的边值问题。以他的名字命名的George Polya奖是为了那些出色地应用了组合理论的人而设立的。

然而，在George Polya为数学做出的所有贡献中，他最自豪的也是让人们记忆最深刻的是他为数学教育所做的贡献。他于1945年出版的著作《如何解决它》销量过百万，而且被翻译成了17种语言。在这本书中，Polya概述了为数学问题设计的问题求解策略。但是，这个策略具有通用性，适用于求解各种问题。Polya的策略正是本书概述的计算机问题求解策略的基础。1954年出版的《数学和似真推理》（Mathematics and Plausible Reasoning）是他献给数学教育的另一本著作。他不仅为数学教育撰写书籍，还醉心于数学教学。他是海湾地区的学校的常客，而且访问过西部各州的大多数大学。爱达荷大学的数学中心就是以他的名字命名的。

1985年9月7日，George Polya在Palo Alto逝世，享年97岁。

寻找熟悉的情况

永远不要彻底重新做一件事。如果解决方案已经存在了，就用这种方案。如果以前曾经解决过相同或相似的问题，只需要再次使用那种成功的解决方案即可。通常，我们意识不到“我以前见过这种问题，我知道该如何处理它”，而只是苦苦求索。人类是擅长识别相似的情况的。我们根本不必学习如何去商店买牛奶，然后买鸡蛋，再买糖果。我们知道，去商店购物这件事都是一样的，只是买的东西不同罢了。

识别相似的情况在计算领域内是非常有用的。在计算领域中，你会看到某种问题不断地以不同的形式出现。一个好的程序员看到以前解决的任务或者任务的一部分（子任务）时，会直接选用已有的解决方案。例如，找出一个温度列表中每天的最高温和最低温与找出一个测验列表中的最高分和最低分是完全相同的任务，想得到的不过是一个数字集合中的最大值和最小值而已。

分治法

通常，我们会把一个大问题划分为几个能解决的小单元。打扫一栋房子或一个公寓的任务看起来很繁重。而打扫起居室、餐厅、厨房、卧室和浴室的独立任务看起来容易多了。这项原则尤其适用于计算领域。把大的问题分割成能够单独解决的小问题。

这种方法应用了第1章中讨论的抽象概念，打扫房子是个大的、抽象的问题，它由打扫每个房间这些子任务构成。打扫房间仍然可以被看作是一种抽象，它的子任务更加详细，如叠

衣服、铺床和给地板吸尘等。可以把一项任务分成若干个子任务，而子任务还可以继续划分为子任务，如此进行下去。可以反复利用分治法，直到每个子任务都是可以实现的为止。

6.1.2 应用Polya的问题求解策略

下面，让我们应用这些策略（称为*探索性方法*）来解决一个特定的问题，即如何到达下个星期六在Sally家举行的聚会。

问题：Sally家在哪里？我们从哪里出发？天气如何？我们走路去？还是开车去？抑或乘坐公共汽车去？这些问题都得到解答后，你就可以开始设计解决方案了。

如果那天下雨，汽车还在商店中，公共汽车停开了，那么最后的解决方案可能是打电话叫出租车，告诉司机Sally家的地址。

如果我们自己开车，查阅地图后知道Sally家在我们公司大厦的西边，相隔6个街区，那么解决方案的第一部分可能是重复我们每天早晨都会做的事情——上班（假设我们从家出发）。接下来是从公司出来后向西转，驾车走6个街区。如果我们记不清过了几个街区，那么需要带一支铅笔，每当经过一个街区，就在纸上做一个记号。虽然重复地做记号对人来说显得有些麻烦，但在计算机解决方案中这是很常用的。如果要重复一个处理10次，就必须编写指令，在每次处理结束时记数，并且检查次数是否达到了10。在计算领域，这种处理方法叫做*重复*或*循环*。

有些人是从一个地方出发的，而有些人是从另一个地方出发的，如果我们需要给这些人写说明，就必须编写两套说明，第一个问题都是“从哪里出发”。如果从A出发，则采用第一套说明，否则，采用第二套说明。在计算领域，这种处理方法叫做*条件处理*。

根据一步一步的过程解决特定的问题并非不会发生任何变化。事实上，反复试验的处理通常需要多次尝试和改进。我们将测试每种尝试，看它是否能真正地解决问题。如果它确实能解决问题，就不需要进一步的尝试，否则需要测试其他的尝试。

6.2 算法

Polya列表的第二步中的最后一句说，最终应该得到解决方案。在计算领域，这种解决方案被称为*算法*。算法是在有限的时间内用有限的数据解决问题或子问题的一套指令。这个定义暗示，算法中的指令是明确的。在计算领域中，必须明确地描述人类解决方案中暗含的条件。例如，在日常生活中，我们不会仔细考虑一个总出现的解决方案。此外，如果一个解决方案要求处理的信息量比我们能够处理得多，它也不会被选用。在计算机解决方案中必须明确这些限制，因此算法的定义包括它们。

Polya列表的第三步是执行解决方案，也就是测试它，看它是否能解决问题。第四步是分析解决方案，以备将来使用。

算法（algorithm）：在有限的时间内用有限的数据解决问题或子问题的明确指令集合。

6.2.1 计算机问题求解

计算领域中的问题求解过程包括三个步骤，即*分析和说明阶段*、*算法开发阶段*、*实现阶

段和维护阶段。请参阅图6-2。第一阶段输出的是清楚的问题描述。算法开发阶段输出的是第一阶段定义的问题的通用解决方案。第三阶段输出的是计算机可以运行的程序，该程序实现了这个问题专用的解决方案——算法。除非运行过程中出现了错误，或者需要改变程序，否则第四阶段没有输出。在发生了错误或改变的情况下，它们将被发送回第一阶段、第二阶段或第三阶段。

分析和说明阶段	
分析	理解（定义）问题
说明	说明程序要解决的问题
算法开发阶段	
开发算法	开发用于解决问题的逻辑步骤序列
测试算法	执行列出的步骤，看它们是否能真正地解决问题
实现阶段	
编码	用程序设计语言翻译算法（通用解决方案）
测试	让计算机执行指令序列。检查结果，修改程序，直到得到正确的答案
维护阶段	
使用	使用程序
维护	修改程序，使它满足改变了的要求，或者纠正其中的错误

图6-2 计算机的问题求解过程

图6-3展示了这些阶段之间的交互。粗线标明了各阶段间的一般信息流。细线表示在发生问题时可以退回前面的阶段的路径。例如，在生成算法时，可能会发现问题说明中的错误或矛盾，这样就必须修改分析和说明。同样，程序中的错误可能说明必须修改算法。

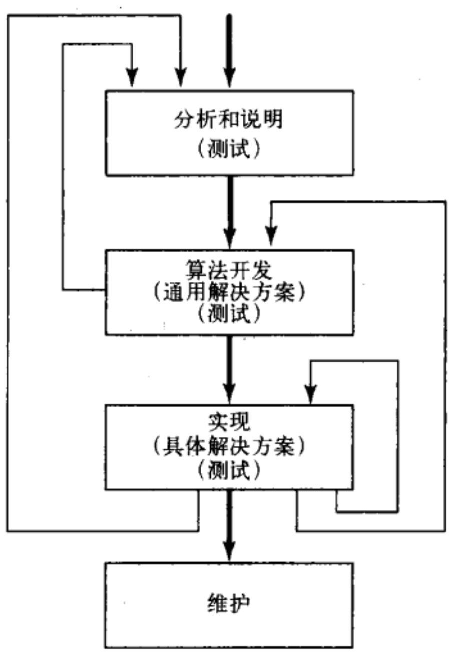


图6-3 问题求解过程中各个阶段的交互

在这个如何用计算机解决问题的略图中，包括Polya列表中的所有阶段。第一步都是理解问题，不可能给根本不理解的问题编写计算机解决方案。接下来是开发解决方案——算法。表示算法的方式有很多种。这里我们介绍一种方法，如下所示，它是把十进制数转换成其他数制的算法。

```
While（商不等于0）
    用新的基数除十进制数
    把余数作为答案最左边的一位数
    用商替换原来的十进制数
```

这种表示算法的形式称为*伪代码*，它利用自然语言和格式明确显示出了解决方案中的步骤。无论采用何种形式表示算法，都必须仔细检测它，确保它确实能解决问题。我们将在6.3节更详细地介绍伪代码。

接下来的步骤是用计算机能够执行的形式实现算法，并且检测结果。在Polya列表中，由人来执行解决方案和评估结果。在计算机解决方案中，程序是用计算机能够执行的语言编写的。但获取程序运行结果，并且检查结果以确保它们正确的则是人。维护阶段对应的是Polya列表中的最后一个阶段，即分析结果，进行必要的修改。

使用计算机进行问题求解只是Polya问题求解方法的一个特例。在计算机问题求解的交互图中，每个阶段都加入了测试这个环节，而不只是在程序运行阶段才进行测试。我们将在本章后面的小节中介绍更多关于测试的内容。

6.2.2　执行算法

虽然你始终都在和算法打交道，但是大部分经验还是*执行*算法。每次按照菜谱做菜、玩游戏、组装玩具或者吃药，都是在执行算法。让我们仔细研究一个菜谱，看看它和算法的描述有哪些吻合之处，如图6-4所示。[2]

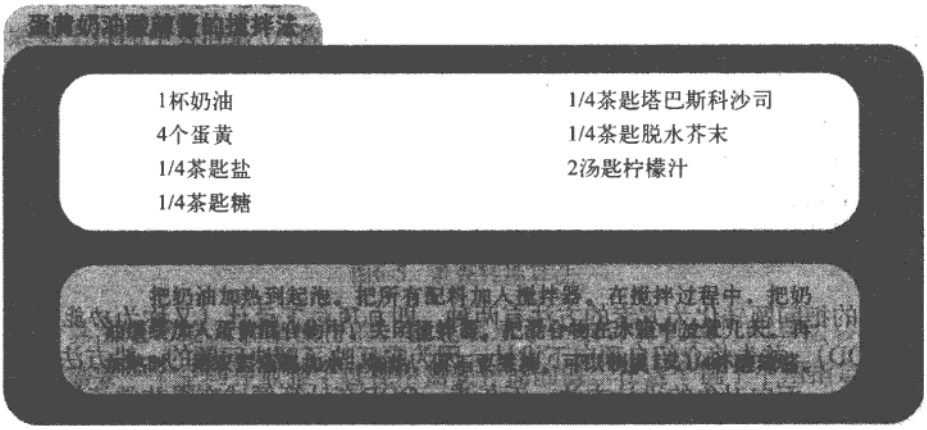


图6-4　蛋黄奶油酸辣酱的菜谱

算法中的步骤一定要能够解决问题。在这个例子中，问题是准备蛋黄奶油酸辣酱，这是一款著名的用于肉排或Benedict蛋的蛋黄酱。这种一步步的说明是可执行的吗？其中的配料是有限的吗？可以在有限的时间内制作出来吗？假设火炉是热的，奶油最终会起泡。那么，所

有问题的答案都是"是的"。这绝对是一个算法。当然，执行算法比设计算法容易。

让我们重新组织这个菜谱，用前面使用的算法格式把它表示出来。

```
If在意卡路里
    把奶油替代品倒入锅内
Else
    把奶油倒入锅内
打开火炉
把锅放在火炉上
While（未起泡）
    把锅置于火炉上
把其他配料放入搅拌器
启动搅拌器
While（还有奶油）
    把奶油缓缓注入搅拌器
关闭搅拌器
```

如第5章中所说的，计算领域有自己的行话，烹饪领域也是如此。"Repeat"和"While"都是在程序设计这个背景中具有意义的术语。"起泡"则是任何厨师都能看懂的术语。*原生词*是指对执行算法的人或设备有意义的术语。上述术语都是原生的。

在菜谱这个比喻中，担当算法开发的另有其人，他们还可能实现算法，包括测试算法。当我们在制作蛋黄奶油酸辣酱时，已经处于维护阶段了，所做的只是在使用别人开发和测试过的菜谱而已。

6.2.3 开发算法

在把问题求解策略应用到问题的同时，我们简短地介绍一下算法开发。我们提出几个问题，考虑几种解决的方法。人们的日常生活与问题求解过程息息相关，太过频繁地进行问题求解，使得它成了人们的一种本能。我们日复一日地重复着Polya列表中的每个步骤，但是却意识不到这一点。只有当遇到重大的问题，需要花费时间来分析在做的是什么时，才会注意到每个阶段之间的过渡。

在计算领域，实现阶段要把解决方案转换成计算机能够执行的形式。为了实现这种转换，需要用适当的形式表示解决方案。因此，必须开发一种方法，以问题陈述开头，以形式转换后的解决方案（算法）结束。本章后面的小节将讨论把算法转换成计算机能够执行的形式——*程序*的过程。

当前使用的把问题转化为方案的方法有两种，即*自顶向下设计*（又称为*功能分解*）和*面向对象设计*（OOD）。首先将介绍自顶向下设计，因为它反映了解决问题的一般方法。之后将介绍比较新的方法——面向对象的设计方法。近年来，OOD在计算领域变得非常流行。这两种方法都是基于分治策略的。

在开始介绍这些方法前，我们先来看看伪代码，即用来表示算法的语言。

6.3 伪代码

首先仔细回顾第2章中把十进制数转换成其他进制数的算法的伪代码。这里将说明计算机

算法和人类执行的算法之间的异同点，然后说明一些在计算机算法的伪代码中一定会用到的结构。

6.3.1 执行一个伪代码算法

```
While（商不为0）
    用新的基数除十进制数
    余数作为结果中的最左边一位
    用商替换原来的十进制数
```

为了帮助我们回忆，我们用这个算法把十进制数93转换成八进制数。用8（新基数）除93，商为11，余数为5。这是第一轮除法，5成为结果的个位数。原始的十进制数（93）将被商11替换。由于商不为0，所以用8除11，商为1，余数为3。数字3成为结果中5左边的数字，临时结果即35。当前的十进制数（11）将被商1替换。由于商不为0，所以继续用8除它，商为0，余数为1。数字1成为结果中最左边的一位数，即结果为135。由于商为0，整个过程结束。

这种解释是正确的，但是却令人费解。让我们从头开始，给需要存储的值一个名字，即decimalNumber、newBase、quotient、remainder和answer。我们用指定的方框来描述这些数据项，方框中是它们的值。如图6-5a所示。对于内容未知的方框，其中放置的是问号。

算法从询问quotient是否为0开始。现在假设它不为0，后面再讨论这一点。图6-5b展示的是第一次循环后（即8除93）的结果。由于商为11，所以要重复这一过程。图6-5c展示的是这次循环后的值。由于商不为0，所以用8除1，结果显示在图6-5d中。现在商为0，整个过程结束。

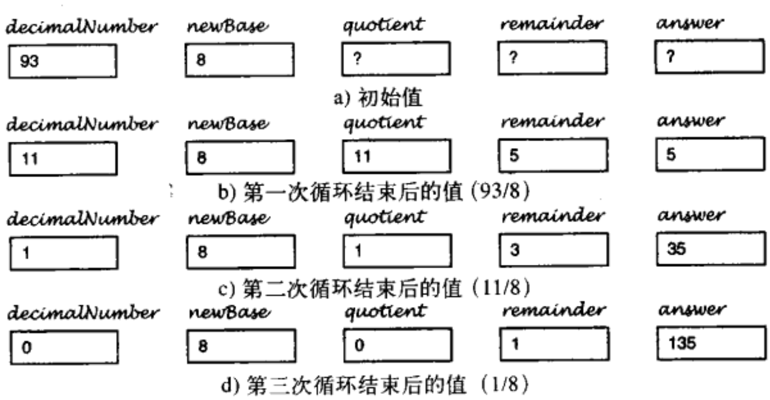


图6-5　走查转换算法

虽然这个算法是写给人们来执行的，但它展示了一些计算机执行的算法必须遵守的重要原则。首先，需要一个空间存放整个过程中需要存储的值。在这个例子中用指定的方框来存储这些值，它们的名字描述了它们在整个过程中的角色。当它们的值发生变化时，新的值会被写入这些方框。在计算机算法中，这些方框叫做变量，可以用它们来存储值，或从中提取值。

其中一个方框decimalNumber最初存放的是这个问题的初始值，即要转换的数。在计算机算法中，必须给出提示，要求某人通过键盘输入这个值。文本框newBase在整个过程中都没有改变，但是它也需要从键盘输入，因为这个算法就是要把十进制数转换成另一种基数的值，所以必须给这个问题输入新基数（在这个例子中是8）。

在开始走查这个算法时，我们知道quotient尚未开始计算，所以可以假设它不为0。在计算机执行的算法中，则必须确保商不为0，所以必须在算法的开头把它设置为非0的值。

下面把该算法重写成计算机可以执行的算法。DIV是一个操作符，返回的是十进制的商；REM也是一个操作符，返回的是十进制的余数。

```
Write "Enter the new base"
Read newBase
Write "Enter the number to be converted"
Read decimalNumber
Set quotient to 1
While (quotient is not zero)
    Set quotient to decimalNumber DIV newBase
    Set remainder to decimalNumber REM newBase
    Make the remainder the next digit to the left in the answer
    Set decimalNumber to quotient
Write "The answer is "
Write answer
```

6.3.2 伪代码的功能

在第1章中我们讲过，语言层包围着真正的机器。当时并未提及伪代码，因为它并非一种计算机语言，而更像一种人们用来说明操作的便捷语言。虽然伪代码并没有特定的语法规则，但必须要表示出下面的概念：

变量 出现在伪代码算法中的名字，引用的是存储值的位置（文本框）。这些名字要能反映出它存放的值在算法中的角色。

赋值 如果有了变量，就要有把值放入变量的方法。可以采用下面的语句：

```
Set quotient to 1
```

这个语句把一个值存放到了变量（方框）quotient中。另一种表示同一概念的方法是使用反向箭头（←）：

```
quotient <— 1
```

如果用赋值语句把值赋给变量，那么之后如何访问它们呢？可以用下面的语句访问decimalNumber和newBase中的值：

```
Set quotient to decimalNumber DIV newBase
```

或

```
quotient <— decimalNumber DIV newBase
```

存放在decimalNumber中的值被存放在newBase中的值除。因此，当变量用于“to”或者“←”右边时，就能访问它存储的值。当变量用于“Set”后或“←”的左边时，就会向其中存入一个值。

存入变量的值可以是单个的值（如1），也可以是由变量或操作符构成的表达式（如decimalNumber DIV newBase）。

输入/输出 大多数计算机程序只处理某种类型的数据，所以必须能够从外部世界向计算

机中输入数据值，还要能把结果输出到屏幕上。在上一节中，算法使用“Write”语句进行输出，使用“Read”语句进行输入。

```
Write "Enter the new base"
Read newBase
```

双引号之间的字符叫做字符串，它们告诉了用户要输入什么或者要输出什么。究竟采用Display还是Print，是无关紧要的，它们都等价于Write，Get和Input都与Read同义。记住，伪代码算法是写给人们看的，以便之后可以把它转换成程序设计语言。不仅对于你自己，还对于要理解你所写的算法从而把它转换成程序的其他人来说，在一个项目中保持使用一致的单词是一种好习惯。

最后两个输出语句说明了重要的一点：

```
Write "The answer is "
Write answer
```

第一条语句把双引号之间的字符输出到屏幕上。第二条语句把变量answer中的内容输出到屏幕上。answer中的值并未改变。

重复 使用重复结构可以重复执行指令。如我们在转换算法中见到的，While语句指示只要条件没有满足，就重复执行算法的某一部分。在我们的示例中，重复执行的部分被缩进了，这与用大括号括起来是一样的。控制重复是否继续的条件通常显示在While旁边的括号中。在我们的示例中，条件是（quotient is not zero）。只要商不为0，重复就会继续。

下面的例子将读入和输出5个数：

```
Set repetitions to 0
While (repetitions < 5)
    Read number
    Print number
    Set repetitions to repetitions + 1
```

重复又叫做循环或迭代。第8章将介绍几种不同类型的循环，不过本章只用While循环。

选择 用选择结构可以选择执行或跳过某项操作。此外，用选择结构还可以在两项操作之间进行选择。与重复结构一样，选择结构使用括号中的条件决定执行哪项操作。例如，下面的伪代码段将读入一个数，如果它大于0，就输出它：

```
Read number
If (number > 0)
    Print number
// whatever statement comes next
```

同样，这里使用缩进对代码段进行分组（在这个例子中只有一组）。控制流将返回到没有缩进的语句。符号“//”用于加注释，它并不是算法的一部分。下面的代码段将输出一个数是正数还是负数：

```
Read number
If (number > 0)
    Print number
    Print " is positive."
```

```
Else
    Print number
    Print " is negative or 0"
```

第一个版本叫做if-then版本，第二个版本叫做if-then-else版本。

表6-1总结了这些语句，并且展示了相关的示例和每种语句使用的关键词。

表6-1　伪代码语句

结　构	含　义	关键词或示例
变量	表示存储变量值或从中提取变量值的指定位置	在伪代码中表示一个值在问题中的角色的名字
赋值	把值存入变量	Set number to 1 number <— 1
输入/输出	输入：读入一个值，可能是从键盘读入的 输出：显示一个变量的内容或一个字符串，可能显示到屏幕上	Read number Get number Write number Display number Write "Have a good day"
重复（迭代、循环）	只要条件没有满足就重复执行一条或多条语句	While (condition) 　//执行缩进的语句
if-then选择	如果条件满足，就执行缩进的语句；如果条件不满足，就跳过缩进的语句	If (newBase = 10) 　Write "You are converting " 　Write " to the same base." //其余代码
if-then-else选择	如果条件满足，就执行缩进的语句；如果条件不满足，则执行Else之后的缩进语句	If (newBase = 10) 　Write "You are converting " 　Write " to the same base." Else 　Write "This base is not the " 　Write "same." //其余代码

这四种基本操作（输入/输出、赋值、重复和选择）构成计算机程序的基础。

6.3.3　伪代码示例

让我们读入一些正数数对，然后按序输出这些数对。如果数对多于一对，就必须使用循环。下面是该算法的初稿：

```
While (more pairs)
    Write "Enter two values separated by a blank; press return"
    Read number1
    Read number2
    Print them in order
```

如何知道何时停止呢？也就是说，如何终止出现更多的数对？可以要求用户告诉程序要输入多少个数对。下面是算法的第二稿：

```
Write "How many pairs of values are to be entered?"
```

```
Read numberOfPairs
Set numberRead to 0
while (numberRead < numberOfPairs)
    Write "Enter two values separated by a blank; press return"
    Read number1
    Read number2
    Print them in order
```

如何判断数对的顺序呢？可以用条件结构比较它们的值。如果number1小于number2，则先输出number1，再输出number2。否则，就先输出number2，再输出number1。在完成算法前，是不是忘了什么呢？numberRead的值从未改变过，必须增加numberRead的值。

```
Write "How many pairs of values are to be entered?"
Read numberOfPairs
Set numberRead to 0
while (numberRead < numberOfPairs)
    Write "Enter two values separated by a blank; press return"
    Read number1
    Read number2
    If (number1 < number2)
        Print number1 + " " + number2
    Else
        Print number2 + " " + number1
    Increment numberRead
```

语句

```
Print number1 + " " + number2
```

中的加号是follow by的便捷符号。number1的内容后是空白字符串，然后是number2的内容。

在没有测试之前，算法都不算完成。可以采用模拟基数转换算法的方法来测试算法。可以选择数值，用纸和笔进行代码走查。这个算法有四个变量需要跟踪，即numberOfPairs、numberRead、number1和number2。假设用户输入了下面的数据：

```
3
10 20
20 10
10 10
```

图6-6a显示了循环开始时各个变量的值，numberRead小于numberOfPairs，所以进入循环。显示提示信息，读入两个数。number1是10，number2是20，所以if语句执行then分支，输出number1，之后是number2。给numberRead加1。图6-6b显示的是第一次重复结束时的值。numberRead仍然小于numberOfPairs，所以重复执行这段代码。显示提示信息，读入两个数。number1是20，number2是10，所以执行else分支，输出number2，之后是number1。给numberRead加1，图6-6c显示的是第二次迭代结束时变量的值。

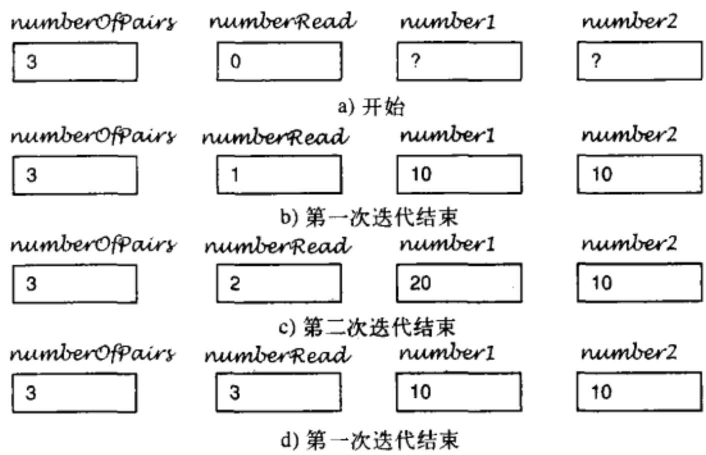


图6-6　数对算法的走查

numberRead小于numberOfPairs，所以重复执行这段代码。显示提示信息，读入两个数。number1是10，number2是10。由于number1不小于number2，执行else分支。输出number2，之后是number1。由于两个值相同，所以它们的输出顺序无关紧要。给numberRead加1。现在numberRead不再小于numberOfPairs，所以代码不再重复。

这一过程叫做*桌面检查*。我们坐在桌子前，用纸和笔走查整个设计。在推理算法时，采用真实的数值来跟踪发生的情况非常有用。这种方法虽然简单，但却极其有效。我们将在本章后面的部分继续介绍测试。

迄今为止，我们已经开发了一个小算法，并且验证了它是正确的。既然已经有了编写操作执行的工具，那么如何决定要执行哪些操作呢？也就是说，如何编写算法呢？下一节将介绍两种不同的设计方法，即自顶向下的设计方法和面向对象的设计方法。

6.4　自顶向下设计方法

自顶向下设计最初把问题分解成一套子问题，然后再把子问题分解成子问题。这一过程将一直持续到每个子问题足够基础，不再需要进一步分解为止。我们创造了一种分层结构来表示问题和子问题（称为**模块**）之间的关系，这种结构也称为*树形结构*。在树形结构中，每一层中的模块都可以调用下层模块的服务。这些模块是算法的基本构件，也正是上一节中所做的。

> **自顶向下设计**（top-down design）：一种程序开发技术，其中问题被分解为更容易处理的子问题，这些子问题的解决方案组合起来构成整体问题的解决方案。
>
> **模块**（module）：一个用于解决问题或子问题的封闭步骤集合。

把问题分解成子问题、模块或者片段的目的，是要独立地解决每个模块。在计算领域中，一个模块可能用于读取数据，一个模块可能用于对数据求和，一个模块用于输出所求的和，而另一个模块则用于比较上一周得到的总值和当前所求的和。

树形结构中包括细化的后继层（如图6-7所示）。0层是最顶层，这一层是问题的功能说明，其下是细化的后继层。那么，如何把问题分解成模块呢？

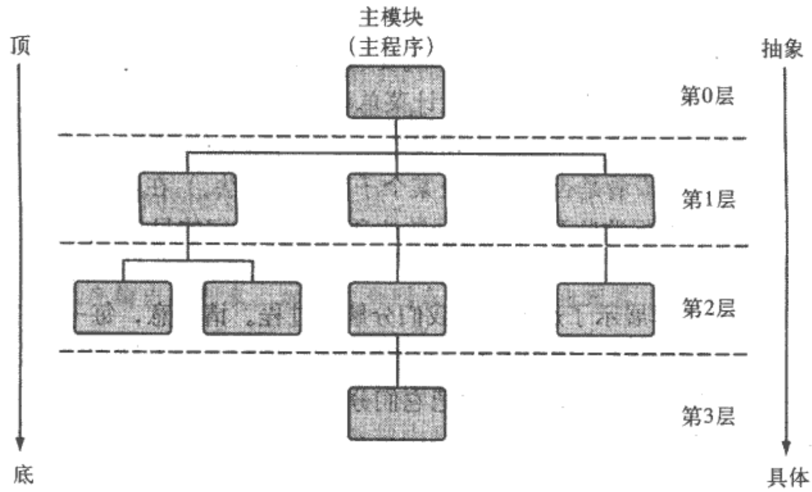


图6-7 自顶向下设计的例子

请想想看人通常是如何解决大问题的。一般，我们会花些时间从全局考虑一下问题，然后简单记下主要步骤，再分析每个步骤，填充它的细节。如果我们不知道如何完成某个任务，就先执行下一个任务，当得到更多的信息后，再返回执行这个跳过的任务。我们所做的是什么呢？是使用分治策略，把问题分解成子问题。

这正是设计算法时要使用的过程。写下主要步骤，它们将成为主要模块。然后开始开发第一层模块中的主要步骤的细节。如果还不知道如何解决一个步骤，或者觉得细节问题很棘手，那么给这个模块起一个名字，继续开发下一个模块。之后，可以把这个名字扩展成低级的模块。

这个过程将在多个层次中重复，把每个任务扩展成最小的细节。需要扩展的步骤是**抽象步骤**，不需要扩展的步骤是**具体步骤**。如果一个任务困难繁重，可以把它的细节推到较低的层次中。这一过程也可以应用到棘手的子任务上。整个问题最终将被分解成能够解决的单元。

> **抽象步骤**（abstract step）：细节仍未明确的算法步骤。
> **具体步骤**（concrete step）：细节完全明确的算法步骤。

编写自顶向下的设计方案与编写论文的大纲相似。尽管计算还是一个新领域，但是这种解决问题的方法却是现实生活中常用的。

6.4.1 一个通用的实例

让我们用这种自顶向下的设计方法处理一个令人愉快的任务——筹划一个大型聚会。简单想一下，可以发现两个主要任务，即邀请参加聚会的人和准备食物。（这个例子忽略了打扫房子这个任务。）

邀请人的方法之一是找到电话簿，开始给朋友打电话。但是，哪些人是已经通知过的？谁的电话占线？给谁的留言机上留了消息？谁说过什么？很快我们就会被搞糊涂了。比较好的方法是列出我们想邀请的人的名单，然后把这个名单放在一边，第二天检查一下它，看看哪些人被忘掉了。

在这个可以查阅的名单中加入电话号码。现在，开始打电话，在上面记下所留的消息和答复。要通知到每个人可能需要一些时间，不过我们知道已经进行到了哪一位。当我们估计

出有多少人参加聚会时，就可以开始计划准备食物了。

如果只是冲进厨房就开始烹调食物，那就要靠天来帮你了。预先没有筹划好的工作，会让你措手不及。让我们把这个任务分解成设计菜单和准备食物。

如果借鉴前人的经验，看看烹饪书中建议的菜单，可以节约不少时间和工作。（在计算领域，我们可以查阅文献，看看是否存在解决某个子问题的算法。）在选择菜单时，可以把研究菜谱的工作推后，当准备好购物列表后，再做这项工作。我们的目标是把细节问题推延到适当的时间再解决。

图6-8中的树形结构图展示了迄今为止我们分解的过程。请注意，每一层中的一个模块扩展了上层中的一个步骤或任务。作为人类，我们可以接受第2层中的模块的任务，根据它们的说明执行这些任务。在计算领域，则必须把它们分解成更细致的模块。例如，必须把“写下名字”这个任务分解成下列的子任务：

准备好纸了吗？
　　没有，则准备纸。
准备好笔了吗？
　　没有，则准备笔。
拿起笔。
用笔在纸上写字。
及其他。

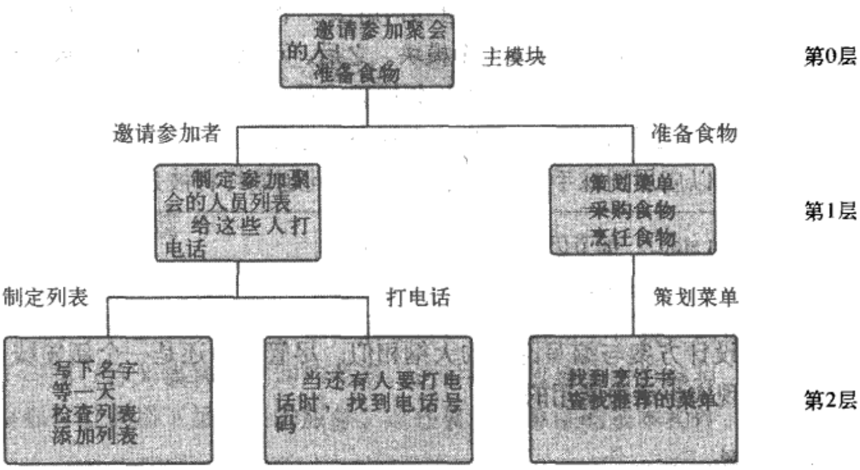


图6-8　划分聚会计划

对聚会的自顶向下设计可能有很大的不同。如果我们所在的街区有家不错的小熟食店，那么可以让他们备办食物。那么主要的模块就变为：

邀请参加聚会的人
打电话给熟食店

总而言之，主模块确定了任务的名字，除非一个任务已经明确了，否则这个任务名需要在下一层中扩展，每一层都是如此。第0层中有多少没有完全明确的任务名，第1层中就有多少模块，对后继层也是如此。

6.4.2 一个计算机实例

让我们分析一个计算机能够解决的问题的处理过程。可以用伪代码来表示算法。

前面提到过，可以用计算机来制定一个列表。让我们把散落在各处的小纸片和名片收集起来，创建一个包括姓名、地址、电话号码和电子邮件地址的列表，列表中的数据以姓的字母顺序排列。第一步要对问题做出明确的陈述。

问题：*创建一个包括每个人的姓名、地址、电话号码和电子邮件地址的地址列表。然后按照字母顺序输出该列表。加入这个列表的姓名是出现在小纸片和名片上的。*

令人意外的是，解决计算机问题的着手点是自问如何手动地解决这个问题。为什么？因为如果我们不能手动地处理一个任务，对这个任务的理解就不够透彻，不足以编写算法。在这个例子中，第一步应该是从各个口袋和钱包中找出所有小纸片和名片，然后坐在桌前整理它们。

手动操作的下一个步骤是在写字板上简单记下每个名字。如果电话号码和电子邮件地址与姓名是一起出现的，那么需要把这些信息加入列表。记下了我们拥有的所有信息后，需要查看是否有遗漏的信息。当所有信息都出现在写字板后，按照字母顺序排列姓名。通常，手动解决方案是计算机解决方案的雏形。在计算机解决方案中，使用的不是写字板，而是把信息输入存放在计算机中的一个列表。现在可以编写代替写字板中的列表的主模块了。

```
Main                                          Level 0

Enter names and numbers into list
Put list into alphabetical order
Print the list
```

现在，必须进一步明确第一个任务——把name输入列表。如前面所说的，人们必须预先收集信息。输入name的过程需要用户键入列表中的数据。可以采用交互模式实现这一过程，即计算机提示需要输入的信息（数据），用户根据提示键入这些信息，也可以预先输入这些数据，由计算机从硬盘读取它们。在这个例子中，我们采用交互式输入数据的算法。

```
Enter names and numbers into list             Level 1

While (more names)
    Enter name
    Enter telephone number
    Enter email
    Insert information into list
```

现在有两个选择，即分解第0层中的第二个任务，或者分解第1层中的第一个任务。我们继续深入树形结构。也就是说，在开始分析如何把列表排序这个任务前，先完成输入数据的操作。如何知道是否还有更多的姓名呢？可以要求用户从键盘输入。在循环之外（第一次迭代）把moreNames设为true，然后在循环结束时询问用户是否还想继续，如果需要，就重置moreNames。首先重写这个模块以反映这种设计决定。

```
Enter names and numbers into list (revised)   Level 1

Set moreNames to true
While (moreNames)
```

```
Prompt for and enter name
Prompt for and enter telephone number
Prompt for and enter email
Insert information into list
Write "Enter a 1 to continue or a 0 to stop."
Read response
If (response = 0)
    Set moreNames to false
```

Prompt for and enter name Level 2

```
Write "Enter last name; press return."
Read lastName
Write "Enter first name; press return."
Read firstName
```

Prompt for and enter telephone number Level 2

```
Write "Enter area code and 7-digit number; press return."
Read telephoneNumber
```

Prompt for and enter email address Level 2

```
Write "Enter email; press return."
Read emailAddress
```

既然有了姓名和相关的数据，就可以把数据插入列表了。还需要进一步改进Insert information into list这个模块吗？答案是可能需要，也可能不需要，这是由把算法转换成程序所采用的语言决定的。我们将在本章后面的小节中讨论转换算法采用的语言。对于这个示例，假设到这一层已经足够详细了。

下一个模块是Put list in alphabetical order。用计算术语来说，按照字母顺序排放列表叫做对列表**排序**。如果列表的内容是数，那么将按照数字顺序进行排序。如果列表的内容是字符串，像这个示例一样，那么将按照字母顺序进行排序。排序算法有很多，我们可以只选择其中的一种（第9章将介绍更多有关排序算法的内容）。首先必须指定要使用哪个域作为**排序关键字**。如果要排序的数据由多个域构成，排序关键字就是我们想以之为依据排列列表的域。在这个示例中，排序关键字是姓名，即firstName和lastName。

> **排序**（sorting）：把列表中的条目按照数字顺序或字母顺序排列。
> **排序关键字**（sort key）：用于排序的域。

最后一个模块是Print the list。这个任务要求遍历列表，输出每个条目。

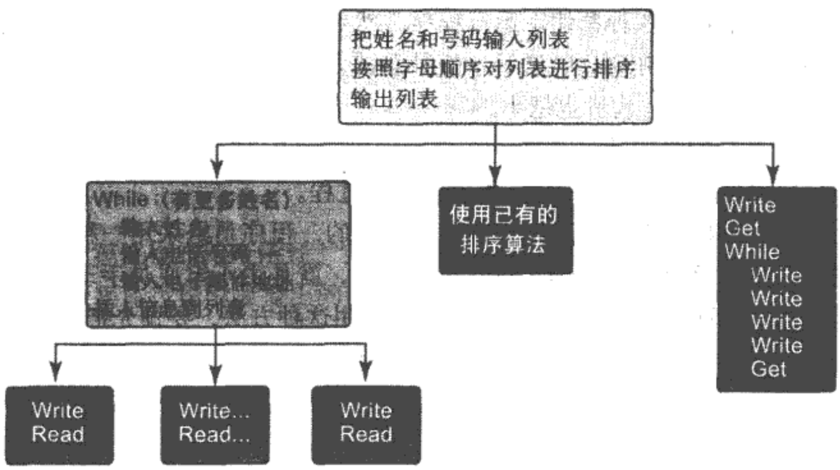
Print the list Level 1

```
Write "The list of names, telephone numbers, and email addresses
follows:"
Get first item from the list
While (more items)
    Write item's firstName + " " + lastName
```

Write item's telephoneNumber
Write item's emailAddress
Write a blank line
Get next item from the list

如何实现条件more items呢？这是由列表的实现决定的，所以只有当把算法转换成语言编写的代码时才能确定这一步。

下面是这个算法的树形结构。模块中黑体文字所表示的任务是需要在下一层中进一步明确的。



6.4.3 方法总结

自顶向下的方法可以分解为四个主要步骤：

分析问题

首先要理解问题，列出必须处理的信息。这些信息可能是问题中的数据。明确采用什么样的解决方案。如果是报表，明确要采用的格式。列出你对问题或信息的假设。思考如何手动地解决这个问题。开发一个全面的算法或通用的方案。

编写主要模块

用自然语言或伪代码在主模块中重述问题。用模块名把问题分解成功能区块。如果主模块太长，说明这一层中的细节太多了。此时可以引入一些控制结构。如果必要，可以进行逻辑重组，把细节推延到下一层模块。

如果目前你不知道如何解决未成文的模块，不必担心。假装你有个非常聪明的朋友知道答案，把这个问题推迟到以后再解决即可。在主模块中所要做的，只是给下一层中每个解决任务的模块一个名字，要采用含义明确的标识符。

编写其余的模块

解决方案中的层数并不确定。每一层中的模块可以载入多个下层模块。虽然上层模块引用的是未成文的下层模块，但上层模块必须完整。不断细化每个模块，直到模块中的每条语句都是具体的步骤为止。

根据需要进行重组和改写

为变化做好打算。不要害怕从头来过。一些尝试和细化操作是必要的。要维持透明性，简单直接地表示你的想法。

6.4.4　测试算法

数学问题求解的目标是生成问题的特定答案，因此，检查结果等价于测试推出答案的过程。如果答案是正确的，过程就是正确的。但是，计算机问题求解的目标是*过程*。嵌在程序中的过程将被反复应用到不同的数据，因此过程自身必须经过测试或验证。

程序的测试通常都是在各种条件下运行程序，然后分析结果以发现问题。不过，这种测试只能在程序完成或至少部分完成时进行，这种测试太迟了，所以不能依赖。越早发现和修正问题，解决问题就越容易，代价也越小。

显然，需要在更早的阶段执行测试。在设计阶段定义的算法必须在实现之前进行测试。这个过程叫做**桌面检查**。在上一节中已经展示了这个过程。

大多数专业计算机程序都是由程序员小组开发成的。程序员组采用一种与桌面检查类似的方法进行程序验证，这种方法称为**走查**。所谓走查，就是由小组成员手动地模拟设计，采用的是实例数据。另一种面向小组的方法是**审查**。使用这种方法，要预先把设计分发给大家，由一位（非设计者）逐行读出设计，而其他人负责指出其中的错误。这些动作是在无胁迫的情况下执行的，目的不是为了批评设计或设计者，而是去除产品中的缺陷。有时，要除去这个过程中人类自然的自负感有些困难，不过最佳的团队都采用了*无我程序设计*的策略。

> **桌面检查**（desk checking）：在纸上跟踪一种设计的执行情况。
> **走查**（walk-through）：由一个小组手动地模拟一种设计。
> **审查**（inspection）：由团队成员之一逐行读出设计，其他成员负责指出错误的验证方法。

接下来的两章会继续讨论“测试”。

6.5　面向对象方法

前面说过，之所以先介绍自顶向下设计，是因为它更能反映人们解决问题的方式。如你所见，自顶向下的解决方案对任务进行了分层。每个任务或指定的动作操作特定的数据来生成想要的输出。自顶向下设计的重点就是任务。相反地，面向对象的设计方法是用叫做*对象*的独立实体生成解决方案的问题求解方法，对象由数据和处理数据的操作构成。面向对象设计的重点是对象以及它们在问题中的交互作用。一旦收集到了问题中所有的对象，它们就能构成问题的解决方案。

6.5.1　面向对象

在面向对象的思想中，数据和处理数据的算法绑定在一起，因此，每个对象负责自己的处理（行为）。面向对象设计（OOD）的底层概念是*类*（class）和*对象*（object）。

对象是在问题背景中具有意义的事物或实体。例如，如果一个问题与学生信息有关，那么在解决方案中，学生就是一个合理的对象。**对象类**（或简称为**类**）描述了一组类似的对象。虽然没有两个学生是完全相同的，但是学生会具有一些共同的属性和行为。学生是（至少大

部分时间）在学校上学的男人或女人。因此，学生可以构成一个类。类这个词指的是把对象归入相关的组，描述它们共性的思想。因此，类描述的是类中的对象表现出的属性和行为。特定的对象只是类的一个实例（具体的例子）。

面向对象的问题求解方法需要把问题中的类隔离开来。对象之间通过发送消息（调用其他对象的子程序）进行通信。类中包含的**域**表示类的属性和行为。**方法**是处理对象中的数据的指定算法。一般说来，类是一种模式，说明了对象看来像什么（数据）以及它的作用（方法）。

> **对象**（object）：在问题背景中相关的事物或实体。
> **对象类**（object class）或**类**（class）：一组具有相似的属性和行为的对象的描述。
> **域**（fields）：类中的特定项，可以是数据或子程序。
> **方法**（method）：定义了类的一种行为的特定算法。

6.5.2 设计方法

我们提出的分解过程有四个阶段。集体讨论是确定问题中的类的第一个阶段。在过滤这个阶段中，将回顾集体讨论阶段提出的类，看哪些类是可以合并的，以及还缺少哪些类。过滤阶段保留下来的类将在下一个阶段仔细研究。

*场景*阶段将确定每个类的行为。由于每个类只对自己的行为负责，所以我们称类的行为为责任。这个阶段探讨的是“如果……将会怎么样”的问题，以确保所有的情况都被分析到了。当每个类的责任都确定后，它们将被记录下来，同时记录的还有它们必须与之协作（交互）才能履行责任的类的名字。最后是*责任算法*阶段，这个阶段将为列出的所有责任编写算法。CRC卡就是用来记录这一阶段的类信息的工具。

<table>
<tr><td>类名：</td><td colspan="2">超类：</td><td>子类：</td></tr>
<tr><td colspan="2">责任</td><td colspan="2">协作</td></tr>
<tr><td colspan="2"></td><td colspan="2"></td></tr>
<tr><td colspan="2"></td><td colspan="2"></td></tr>
<tr><td colspan="2"></td><td colspan="2"></td></tr>
<tr><td colspan="2"></td><td colspan="2"></td></tr>
</table>

让我们详细地看看每个阶段。

集体讨论

什么是集体讨论？字典把它定义为一种集体问题求解的方法，包括集体中的每个成员的自由发言。[3]提到集体讨论，会让人联想到一个电影或电视剧场景，一组生气勃勃的年轻人在毫无拘束地大谈自己关于最新产品的广告标语的创意。而计算机分析员给人们的印象就是独自在封闭的、没有窗户的办公室中工作一整日，最后跳起来大喊“啊哈！”，很难把这种景象和集体讨论联系在一起。随着计算机变得越来越强大，能够解决的问题也变得越来越复杂，这种把自己锁在没有窗户的房间中的场景已经过时了。复杂的问题需要集思广益，以得到具有创新性的解决方案。

在面向对象的问题求解方法中，集体讨论是一种集体行为，为的是生成解决某个特定问题要用到的候选类的列表。在进行广告标语的集体讨论之前，所有参加者都要先了解产品，同样地，在对类进行集体讨论前，参加者也必须了解问题。每个进入集体讨论会议的成员，都应该清楚地理解了要解决的问题。毫无疑问，在准备过程中，每个成员都会草拟出自己的类列表。

虽然集体讨论是一项集体活动，但你可以针对小问题自行锻炼。

过滤

集体讨论会生成一份暂时的类列表。下一阶段要根据这个暂时的列表，确定问题解决方案中的核心类。在这份列表中，也许有两个类其实是相同的。这种类重复的现象很常见，因为一个公司不同部门的人员会对相同的概念或实体采用不同的名字。另外，也许有两个类中有许多共同的属性和行为。应该把共同具有的部分集中在一个超类中，这两个类继承超类中的共同属性，然后再分别附加不同的属性。

在这份列表中，也许有的类根本不属于问题的解决方案。例如，如果我们要模拟一个计算器，那么可能会把用户列为一个可能的类。但用户并不是模拟器中的一个类，用户只是这个问题之外的一个实体，用于给模拟器提供输入而已。另一个可能的类是“开”按钮。但仔细想一下就会发现，“开”按钮也不是模拟器的一部分，它仅用于启动模拟程序。

在完成过滤之后，这个阶段保留下来的所有类将被传递到下一阶段。

场景

这个阶段的目标是给每个类分配责任。最终，责任将被实现为子程序。在这个阶段，我们感兴趣的只是“任务是什么”，而不是“如何执行任务”。

责任的类型有两种，即类自身必须知道什么（知识）和类必须能够做什么（行为），类把它的数据（知识）封装了起来，一个类的对象不能直接访问另一个类中的数据。所谓**封装**，就是把数据和动作集中在一起，使数据和动作的逻辑属性与它们的实现细节分离。封装是抽象的关键。不过，每个类都有责任让其他类访问自己的数据（知识）。因此，每个类都有责任了解自身。例如，学生类应该“知道”自己的名字和地址。使用学生类的类都应该能够“获取”这些信息。这些责任的命名规则通常是“获取”加数据名，例如“获取名字”或“获取电子邮件地址”。了解这种知识的责任被称为“知道名字”和“知道地址”。无论电子邮件地址是保存在学生类中，还是学生类必须通过其他类来获取地址，在这个阶段都是不相干的。重要的是学生类知道自己的电子邮件地址并且能够把它返回给需要它的类。

> **封装**（encapsulation）：把数据和行为集中在一起，使数据和行为的逻辑属性与它们的实现细节分离。

行为的责任看起来更像自顶向下设计中的任务。例如，学生类的一个责任可能是计算它的年级平均成绩（gpa）。在自顶向下设计中，我们会说这个任务是计算特定数据的gpa。在面向对象设计中，我们会说学生类要负责计算自己的gpa。这之间的差别微妙而深奥。尽管最后的计算代码可能看来相同，但是它的执行方式却完全不同。在以自顶向下设计为基础的程序中，将调用计算gpa的子程序，把学生对象作为参数传递。在基于面向对象设计的程序中，将给学生类的对象发送消息来计算gpa。这里没有参数，因为得到消息的对象知道自己的数据。

这个阶段的名字提示了如何给类分配责任。整个小组演出不同的场景。场景是讲述“如

果……将会怎么样”的剧本，使参与者能够把每种情况都表演一次。

这个阶段输出的是一套代表分配了责任的类，可能写在CRC卡上。卡上列出了每个类的责任，以及每个责任需要协作的类。

责任算法

最终必须为责任编写算法。由于在面向对象的设计观念中，重点是数据而不是行为，所以执行责任的算法一般都相当短。例如，知识责任通常只返回一个对象的变量的内容，或者给另一个对象发送消息来检索它。行为责任复杂一些，通常涉及计算。因此，自顶向下设计算法的方法通常也适用于设计责任算法。

总结

让我们做个总结，自顶向下的设计方法重点在于把输入转化成输出的*过程*，结果将生成层次化的任务体系结构。面向对象设计的重点是要转换的*数据对象*，结果生成的是层次化的对象体系结构。Grady Booch用这样的方法分类：“阅读要构造的软件的说明。如果要编写程序性的代码，就用下划线标出动词；如果要编写面向对象的程序，请标出名词。”[4]

我们建议用圆圈圈出名词，用下划线标出动词。名词可以成为对象，动词可以成为操作。在自顶向下设计中，动词是重点；在面向对象设计中，名词是重点。

6.5.3 一个通用的实例

我们首先应用自顶向下的过程来筹划了一个大聚会。现在，让我们用面向对象的过程来重新筹划一下这次聚会。

集体讨论和过滤

这里与一个或两个人进行集体讨论有点困难，所以我们必须用语言模拟集体讨论和过滤。可能的对象有哪些？应该有男主人或女主人、客人、菜单和食物。聚会在哪里举行？需要一个地址对象。聚会何时举行？需要一个日期对象。是打电话邀请客人还是发请帖？假设使用手写的请帖，那么还需要一个请帖对象，由日期和地址对象构成。

如果准备发送请帖，就需要一个邀请的客人的列表。列表是一个*容器*对象，即包含其他对象的对象。为了准备食物，我们还需要知道菜单是什么以及有多少人会来，这意味着需要三个容器对象，一个是邀请的客人的列表，一个是会参加聚会的客人的列表，还有一个食物列表。

可以把一些对象组合到一个类中吗？男主人和女主人与客人没有什么区别，只不过他们要启动这个过程，把客人的名字插入邀请列表中。可以把这些对象组合到一个类吗？假设有一个人的类，男主人、女主人和客人都是这个类的对象。目前，我们假设有一个列表类，它有两个对象，一个是邀请的客人的列表，一个是会参加聚会的客人的列表。右边是这个阶段中可能出现的类的列表。

人
请帖
地址
日期
受邀请的人员列表
要出席的人员列表
食物列表

场景

可能出现的场景有哪些？让我们从发送请帖开始。谁负责发送请帖？男主人或女主人。因此，要把发送请帖作为责任添加到人这个类中。不，不对。让所有人都承担这个责任是不合理的。让我们创建一个与人这个类相似的类，把发送请帖的责任添加到这个类中，男主人或女主人

是这个类的对象。(这种区别叫做继承，第8章会对它进行定义。）向某人发出邀请消息后，他要做出响应。所以邀请必须有一个返回值“yes”或“no”。主人还必须设计菜单，所以“设计菜单”是另一个责任。

如你所见，这个过程与自顶向下的解决方案非常不同。现在让我们看看计算机实例。

6.5.4 一个计算机实例

让我们再重复一次前面例子中使用的问题求解过程，不过这次使用的是面向对象的方法。为了唤起你的记忆，这里重述一次问题：

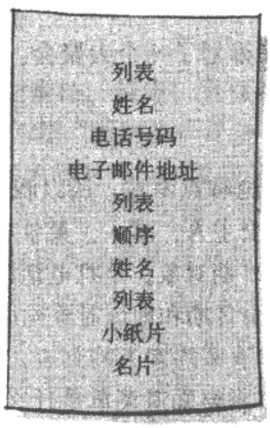
问题：*创建一个包括每个人的姓名、地址、电话号码和电子邮件地址的地址列表。然后按照字母顺序输出该列表。加入这个列表的姓名是出现在小纸片和名片上的。*

集体讨论和过滤

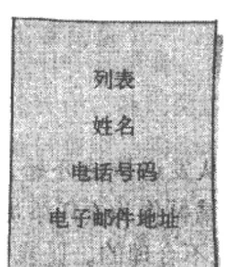
让我们用波浪线标出问题陈述中的名词，用下划线标注动词。

创建一个列表，包括每个人的姓名、电话号码和电子邮件地址。然后按照字母顺序输出该列表。加入这个列表的姓名是出现在小纸片和名片上的。

第一遍列出的类如下所示：



这些类中有三个是相同的，即三个“列表”指的都是创建的容器。*顺序*是一个名词，但*顺序*类是什么呢？实际上，这个名词说明了如何输出列表中的条目，因此我们不把它作为类处理。两个*姓名*类应该组合到一个类中。*小纸片*和*名片*描述的对象包含的是真实世界中的数据。它们在设计中没有与之相对应的类。经过过滤后的类列表如下：



利用问题陈述中的动词，可以顺利地设计出责任，即*创建*、*输出*和*加入*。与小纸片和名

片一样，加入这个动词是让某人准备数据的指令，在设计中没有与之相对应的责任。但是，它说明我们必须具有一个把数据输入列表的对象。要输入的数据究竟是什么呢？它是列表上每个人的姓名、电话号码和电子邮件地址。这一连串的思考让我们发现错过了问题陈述中的一个重要线索。所有格形容词“每个人的”其实提出了一个主要的类，姓名、电话号码和电子邮件地址这些类都是帮助定义（包含在）人这个类的。

现在我们有两个选择。是应该使人这个类具有输入自己的数据来初始化自己的责任，还是应该创建另一个类来输入并把数据发送给人这个类从而初始化它呢？我们选择让人这个类负责自己的初始化，它还要负责输出自己。

人这个类需要和其他类协作吗？这是由如何表示这个类中的数据决定的。是用简单数据类型表示姓名、电话号码和电子邮件地址，还是分别用类表示它们呢？我们用类表示姓名，它具有两个数据项，即姓和名，其他的则用字符串变量表示。姓名类具有知道它的数据值的责任。

类名：Person	超类：	子类：
责任	协作	
初始化自身（name、address、telephone、email）	Name、Address、Telephone、Email	
输出	Name、Address、Telephone、Email	

类名：Name	超类：	子类：
责任	协作	
初始化自身（name）	String	
输出自身	String	

列表对象又如何呢？这个列表是应该按照字母顺序保存条目还是在输出条目时才对它们排序呢？用于实现设计的每种语言都有一个容器类库，让我们采用其中的一个类，这个类是按照字母顺序保存条目的，而且会输出列表。我们可以为这个类创建一个CRC卡，但是要注明它大部分是用类库中的类实现的。

类名：SortedList（来自库）	超类：	子类：
责任	协作	
插入（person）	Person	
输出	Person	

责任算法

Person类　有两个责任需要分解，即初始化和输出。由于姓名是一个类，可以让这些类自己进行初始化并输出自身。在面向对象设计中，调用方法是用对象名加点号再加要调用的方法。

Initialize

name.initialize()
Write "Enter phone number; press return."
Get telephone number
Write "Enter email address; press return."
Get email

Print

name.print()
Write "Telephone number: " + telephoneNumber
Write "Email address: " + emailAddress

Name类　这个类有两个责任，即初始化和输出，它们的算法不同。初始化操作必须提示用户输入姓名，然后读入姓名。输出操作则必须输出姓名，并给出合适的标签。

Initialize

"Enter the first name; press return."
Read firstName
"Enter the last name; press return."
Read lastName

Print

Print "First name:" + firstName
Print "Last name:" + lastName

我们就此停止设计。回顾一下第6章开头介绍的对这个问题的自顶向下设计。这两个设计完全不同。自顶向下的设计是一种以任务为树结点的分级树。面向对象的设计则生成了一组类，每个类有执行自己行为的责任。哪一种更好呢？面向对象的设计更好一些，因为它创建的一些类还可以用于其他背景。*可复用性*是面向对象设计的一大优点。为一个问题设计的类还可以用于解决另一个问题，因为每个类都是自给自足的，也就是说，每个类只负责自己的行为。

6.6　几个重要思想

这一章顺便提到过几个主题，它们不仅对于问题求解重要，在整个计算领域都很重要。让我们回顾一下在这一章中讨论过的一些通用思想。

6.6.1　信息隐蔽

我们使用过几次推延细节的思想。曾经用它先给任务命名，而把如何实现任务推延到以后再考虑。在设计过程中，把细节设计延后具有明显的优势。高层设计隐蔽了细节。对于每

个特定的分层，设计者只考虑与之相关的细节。这种做法叫做**信息隐蔽**，即在进行高层设计时不能见到低层的细节。

这种做法看来非常奇怪。为什么在设计算法时不能见到细节呢？设计者不是应该无所不知吗？不是。如果设计者知道一个模块的低层细节，他或她就可能会以这些细节为基础设计这个模块的算法。但是这些低层的细节很可能会发生变化。一旦它们改变了，那么整个模块都要重写。

> **信息隐蔽**（information hiding）：隐蔽模块的细节以控制对这些细节的访问的做法。

> **濒危物种的管理**
>
> 动物园捕获了一些濒临灭绝的动物，以防止它们灭绝。California秃鹫、Przewalski马和非洲大羚羊都成功地存活了下来。动物园需要对年龄和基因有个好的分类，才能保护不同的物种抵抗疾病以及进行交配。为此建立了一个被捕获动物的计算机数据库以供分析，其中存放了该动物的出生和死亡日期、性别、家系和生活的位置，这样科学家就能衡量哪些因素（例如繁殖和存活率、交配支系以及遗传多样性损失等）才能使该物种受益。

6.6.2 抽象

抽象和信息隐蔽就像一个硬币的两面。信息隐蔽是隐藏细节的做法，抽象则是隐藏细节后的结果。第1章提到过，抽象是复杂系统的一种模型，只包括对观察者来说必需的细节。我们用一种英国狗Daisy做个比喻。对于她的主人来说，她是一只宠物；对于猎人来说，她是一只捕鸟猎犬；对于兽医来说，她是一只哺乳动物。她的主人会看到她摇尾巴，当她要外出时，会听到她的叫声，而且到处都能看到她的狗毛。猎人看到的是一个训练有素的帮手，知道自己的工作，并且能够出色地完成工作。兽医看到的则是构成她身体的器官、血肉和骨头。如图6-9所示。

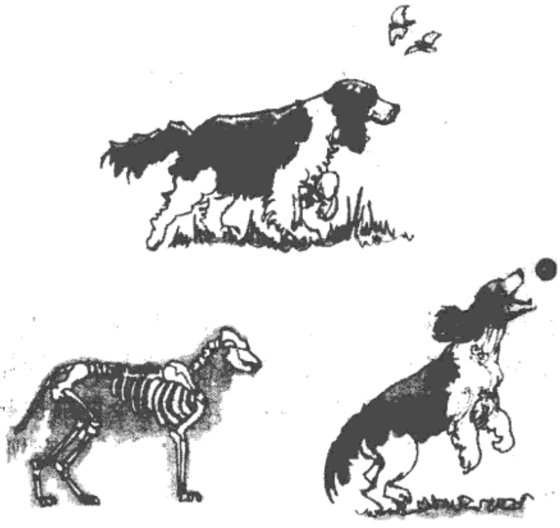


图6-9 同一个概念的不同视点

> **抽象**（abstraction）：复杂系统的一种模型，只包括对观察者来说必需的细节。

在算法设计中，高层的模块是它下面的模块的抽象。在分析把算法转换成程序设计语言时，我们会看到计算领域中的两种主要抽象类型。**数据抽象**指的是数据概观，即把数据的逻辑概观和它的实现分离开。例如，你的开户银行使用的计算机可能是用二进制补码表示数字的，也可能是用一进制补码，但是这种区别对你来说无关紧要，只要你户头中的数字正确即可。**过程抽象**指的是动作概观，即把动作的逻辑概观和它的实现分离开。例如，当你踩下刹车后，车子会停下。踩住刹车是如何使车子停下来的对你来说并不重要，只要车子停下来了即可。

计算领域中的第三种抽象类型叫做**控制抽象**，它指的是*控制结构*的概观，即把控制结构的逻辑概观和它的实现分离开。使用控制结构可以改变算法的顺序控制流。例如，While和If都是控制结构。在用于实现算法的程序设计语言中是如何实现控制结构的，对于算法设计来说并不重要。

抽象是人们用来处理复杂事务的最强有力的工具。这句格言，无论在计算领域，还是在现实生活中，都是适用的。

> **数据抽象**（data abstraction）：把数据的逻辑概观和它的实现分离开。
> **过程抽象**（procedural abstraction）：把动作的逻辑概观和它的实现分离开。
> **控制抽象**（control abstraction）：把控制结构的逻辑概观和它的实现分离开。
> **控制结构**（control structure）：用于改变正常的顺序控制流的语句。

6.6.3 事物命名

在编写算法时，我们使用速记符号表示要处理的任务和信息，也就是说，给数据和过程一个名字，这些名字叫做*标识符*。例如，我们在类name中使用firstName和lastName表示人的姓和名。此外，还要给任务命名。例如，用Get first name表示读取从键盘输入的人的姓。数据值标识符是由单词构成的，用大写字母会使它们更清晰。这里用短语表示任务名。最终任务名仍然必须转换成标识符，如getFirstName。

当我们要用一种程序设计语言把算法转换成计算机能够执行的程序时，可能必须修改标识符。每种语言都有自己构成标识符的规则。因此，转换过程分两个阶段，首先在算法中命名数据和动作，然后把这些名字转换成符合计算机语言规则的标识符。请注意，数据和动作的标识符都是抽象的一种形式。

6.6.4 程序设计语言

我们多次提到“计算机能够执行的语言”这句话。让我们表达得更精确一些。从对冯·诺伊曼机的讨论中我们了解到，CPU一次读取并执行一条指令。计算机能够直接执行的指令是内置在硬件中的指令。但是，用**程序设计语言**编写的指令能够被*翻译*成计算机可以直接执行的指令。程序设计语言是一种人造语言，由符号、专用字和一套规则构成，用于构造**程序**，即表示成计算机能够理解的指令序列。

程序设计语言的形式多种多样，复杂度也不一，但是都由两个部分构成，即语法和语义。**语法**说明了如何组织语言中的指令。**语义**说明了各条指令的含义。

这一章已经介绍了问题求解和算法设计，下一章将介绍如何用程序实现一个算法。把算

法转换成程序的过程叫做*编码*。

> **程序设计语言**（programming language）：用于构造程序（表示成计算机能够理解的指令序列）的规则、符号和专用字集合。
>
> **程序**（program）：用于执行特定任务的指令序列。
>
> **语法**（syntax）：规定有效指令的结构的正式规则。
>
> **语义**（semantics）：赋予程序设计语言中的指令含义的一套规则。

6.6.5 测试

我们已经演示了算法阶段的测试。实现阶段的测试涉及用为测试程序各部分而设计的各种数据运行程序。后面几章将讨论这些阶段的测试理论。但是，我们对设计阶段的测试所做的一切说明都适用于其他阶段。

小结

Polya在他的经典著作《如何解决它》中列出了数学问题的求解策略。这个策略适用于所有问题，包括那些要编成计算机程序的问题。这个策略的步骤是提出问题，寻找熟悉的情况，然后用分治法解决。应用这一策略，将生成一个解决问题的*方案*。在计算领域，这种方案称为*算法*。

伪代码是像人类使用的语言的一种便捷形式，用于表示算法。使用伪代码可以命名变量（存放值的空间）、把数值输入变量以及输出存储在变量中的值。使用伪代码还可以描述算法中重复执行的动作以及选择执行的动作。

人们执行算法（方案）的经验比设计算法多。这一章介绍了自顶向下的设计方法，并且应用它来解决了一个通用问题和一个能够编程的问题。这种方法的基本思想是把一个任务不断地分解成子任务，直到每个子任务的实现方法一目了然为止。这个过程的每个阶段都需要进行测试，以确保结果是正确的。

面向对象设计的重点是确定问题中的对象，并根据对象的属性和行为把它们抽象（分组）成类。下面是面向对象分解的四个阶段。

- *集体讨论*：在这个阶段中，我们为确定问题中的类进行第一轮讨论。
- *过滤*：在这个阶段中，我们将检查提出的类。
- *场景*：在这个阶段中，我们将确定每个类的责任。
- *责任算法*：在这个阶段中，我们将为每个责任编写算法。

这一章提出了四个充斥于整个计算领域的重要思想：信息隐蔽、抽象、事物命名和测试。信息隐蔽是隐藏子任务的细节的过程。抽象是隐藏细节的结果。例如，用车体隐藏汽车的细节，车体就是汽车的抽象。我们会给数据和任务一个名字，以便讨论它们。这些名字就是数据和任务的抽象。在解决与计算机无关的问题时，我们自己执行解决方案，因此知道问题是否解决了。如果问题求解的结果是一个计算机程序，那么就必须全面地测试每个算法，以确保（每次）执行程序后，这个程序给出的答案是正确的。

道德问题：计算机专业人员许可

水管工、电工、美容师、心理医生和专业工程师，这些为大众提供服务的人几乎都要求有许可。会计师（CPA）和特定药学领域的专科医生都要通过认证。但是，计算机专业人员既没有许可，也没有认证。

认证是由专业机构管理的自愿性程序，许可则是政府机关（在美国，通常为州级政府）管理的强制性程序。计算机专业人员的认证协会是软件业建立最完善的认证组织。他们提供两种认证：计算专业协会（Association Computing Professional, ACP）和计算专业认证（Certified Computing Professional, CCP）。这两种认证都要求通过一项测验，该测验包括110个问题，涵盖有关人员和组织架构、系统概念、数据和信息、系统开发和相关原则的主题。ACP认证除了要求通过上述核心测验外，还要通过一项关于程序设计语言的测验。CCP认证除了要求通过核心测验外，还要求通过一项测验，这项测验的主题是在管理、过程程序设计、商业信息系统和系统程序设计中选择两个，此外，参加者还要具有48个月的全职工作经验或者24个月的全职学术活动经验。这项认证倾向于计算的商业应用，而不是计算机专业人员。许多大型商业软件公司（如Apple、Microsoft和Novell）都有自己的认证工具。

有两个主要的专业计算组织——ACM和IEEE计算机协会。这两个组织于1993年成立了软件工程专业建立指导委员会（Steering Committee for the Establishment of Software Engineering as a Profession）。要建立一种专业，指导委员会建议采用标准的定义，定义必需的知识主体及推荐的实践活动，定义道德标准，定义教育课程。1998年，ACM和IEEE联合建立了软件工程协调委员会（Software Engineering Coordination Committee, SWECC），它是一个永久性委员会，目的是把软件工程发展成职业计算的一个分支。得克萨斯大学专业工程许可委员会要求这个委员会协助定义得克萨斯大学管理的软件工程许可测验的评判标准。

医药师、律师和工程师都需要有执照。显然，工程学的模式更适用于计算专业。工程学模型要求候选人具备良好的品质，毕业于公认的工程学专业且具有4年的相关经验，或者没有学位但有8～12年的经验，并且通过了测验。SWECC被要求协助这项测验。ACM是由理事会管理的，理事会成员是由其他成员选举产生的。ACM理事会中的某些成员对许可的软件工程师是否适合某个领域持保留态度。经过进一步的研究，ACM理事会通过了下列提议：

> *ACM反对目前给软件工程师颁发许可，因为ACM相信这项工作还不成熟，对解决软件质量和可靠性的问题没有太大帮助。但是，ACM正在通过促进R&D来解决软件质量的问题，开发了软件工程知识的核心主体，确定了一系列实践的标准。*

ACM反对许可的原因之一是8个小时的工程学基础测验涵盖了工程学学位前两年要学的主题，而其中许多主题（如热力学、水力学、统计学和材料科学）都与计算专业无关。2001年8月，IEEE计算机协会对一个软件工程的认证程序进行了Beta测试。这项认证的要求与许可要求相似，只是其中的学位可以是任何公认高等教育的任何学科的。因此，这项认证的测验不包括一般的工程学主题。

练习

为练习1～10中的活动找到匹配的面向对象设计阶段。

A. 集体讨论　　B. 过滤

C. 场景　　D. 责任算法

1. 浏览可能的类的列表，找出重复或遗漏的类。
2. 提出“如果……将会怎么样”的问题。
3. 给类分配责任。
4. 草拟出问题中的类的第一个列表。
5. 分配责任的协作者。
6. 为一张CRC卡上列出的责任开发算法。

7. 这个阶段输出的是所有类的完整的CRC卡。
8. 这个阶段输出的是准备翻译为程序的OOD。
9. 继承关系是在这个阶段中建立的。
10. 函数程序设计方法适用于这个阶段。

为练习11～16中的叙述找到匹配的定义。

A. 信息隐蔽　　B. 抽象
C. 数据抽象　　D. 过程抽象
E. 控制抽象　　F. 封装

11. 把数据和动作集中在一起，使数据和动作的逻辑属性与它们的实现细节分离。
12. 隐蔽模块的细节以控制对这些细节的访问的做法。
13. 复杂系统的一种模型，只包括对观察者来说必需的细节。
14. 把动作的逻辑概观和它的实现分离开。
15. 把控制结构的逻辑概观和它的实现分离开。
16. 把数据的逻辑概观和它的实现分离开。

练习17～60是设计题或简答题。

17. 请列出Polya提出的“如何解决它”中的四个步骤。
18. 用你自己的话描述练习17中列出的四个步骤。
19. 列出本章讨论的问题求解策略。
20. 把本章讨论的问题求解策略应用于下列情况：
    a) 为你4岁的堂妹买一个玩具。
    b) 为你的足球队组织一场庆功宴。
    c) 为要嘉奖你而举办的宴会买一套礼服或套装。
21. 分析练习20中的解决方案，确定三件事的共性。
22. 什么是算法？
23. 为下列任务编写算法。
    a) 制作花生酱和果酱三明治。
    b) 早晨起床。
    c) 做家庭作业。
    d) 下午开车回家。
24. 列出计算机问题求解模型的三个阶段。
25. 计算机问题求解模型与Polya的模型有哪些不同之处？
26. 描述算法开发阶段的步骤。
27. 描述实现阶段的步骤。
28. 描述维护阶段的步骤。
29. 从烹调书上找一个制作核仁巧克力饼的菜谱，回答下列问题：
    a) 这个菜谱是算法吗？请解释你的答案。
    b) 用伪代码把这个菜谱改编成算法。
    c) 列出在计算领域有意义的单词。
    d) 列出在烹饪中有意义的单词。
    e) 把制作好的核仁巧克力饼带给你的教授品尝。
30. 我们说过，执行菜谱比设计菜谱简单得多。去超级市场买一种你从未做过（或吃过）的蔬菜，为它设计一种菜谱。写下你的菜谱以及对这个过程的评价。（如果是个好菜谱，不要忘记发给本书的作者们。）
31. 描述自顶向下设计的过程。
32. 区分具体步骤和抽象步骤。
33. 为下列任务编写一种自顶向下设计。
    a) 为你4岁的堂妹买一个玩具。
    b) 为你的足球队组织一场庆功宴。
    c) 为要嘉奖你而举办的宴会买一套礼服或套装。
34. 为下列任务编写一种自顶向下设计。
    a) 计算10个测验成绩的平均值。
    b) 计算数量未知的测验成绩的平均值。
    c) 说明这两种设计的不同之处。
35. 为下列任务编写一种自顶向下设计。
    a) 在电话号码簿上找一个电话号码。
    b) 在Internet上找一个电话号码。
    c) 找一个你丢失的小纸片上的电话号码。
    d) 说明这些任务的不同之处。
36. 区分重复和选择。
37. 为按照字母顺序对一个列表中的姓名排序编写自顶向下的设计。
38. a) 为什么信息隐蔽很重要？
    b) 列举三个你每天会遇到的信息隐蔽的例子。
39. 飞机是一种复杂的系统。
    a) 从飞行员的角度给出飞机的一种抽象。
    b) 从乘客的角度给出飞机的一种抽象。
    c) 从空乘的角度给出飞机的一种抽象。
    d) 从维修技工的角度给出飞机的一种抽象。
    e) 从航空公司办事处的角度给出飞机的一种抽象。
40. 列出练习33的设计中的标识符，说明它们命名的是数据还是动作。
41. 列出练习34的设计中的标识符，说明它们命名的是数据还是动作。

42. 列出练习35的设计中的标识符，说明它们命名的是数据还是动作。
43. 什么是“无私程序设计”？
44. 用一些采样数据，对地址列表这个实例进行桌面检查。
45. 用走查的方法验证练习33中的设计。
46. 用审查的方法验证练习34中的设计。
47. 用自顶向下推理的方法验证练习35中的设计。
48. 区别对象和对象类。
49. 区别域和方法。
50. 对象之间有哪些联系？
51. 讨论自顶向下设计和面向对象设计的区别。
52. 我们概述了开发面向对象分解的策略。
    a) 请列出它的四个阶段。
    b) 概述每个阶段的特点。
    c) 每个阶段的输出是什么？
    d) 每个阶段是独立的吗？请解释原因。
53. 用集体讨论、过滤和场景这一策略为汽车经销商设计商品目录系统的CRC卡。
54. 用集体讨论、过滤和场景这一策略为动物园设计数据库的CRC卡。
55. 区分数据抽象和过程抽象。
56. 什么是程序设计语言？
57. 区分语法和语义。
58. 编写伪代码算法，读入一个姓名，然后输出“Good morning”的消息。
59. 编写伪代码算法，读入用户输入的三个整数，然后按照数字排序输出它们。
60. 把练习59中的设计封装在循环中，除非用户输入的三个数中的第一个是负数，否则就不断读入三个数值组。
61. 修改练习59中的算法，如果用户要终止程序，输入一个负数即可（即不需要再输入第二和第三个值）。

思考题

1. 请区分计算机能够直接执行的程序和必须转换的程序。
2. 无论自顶向下设计还是面向对象设计都给编写程序搭建了平台。搭建这些平台都是白费力气吗？当程序完成并且运行时，它们的价值是什么？
3. 你最常用的问题求解策略是什么？能够想出你用过的其他策略吗？它们适用于计算问题的求解吗？
4. ACM反对给软件工程师发放许可。他们的论据是什么？你同意他们的观点吗？为什么？
5. 许可和认证有什么区别？你认为对计算领域内的人士来说，哪种更合适？许可还应该分级吗？认证呢？

第7章 低级程序设计语言

上一章分析了问题求解，包括人类一般如何解决问题以及当解决方案涉及计算机时如何解决问题。这两种情况的第一阶段都是制定方案或算法。在Polya提出的“如何解决它”列表中，由人类执行解决方案并评估结果。在计算机解决方案中，需要编写程序，用一种程序设计语言表示出这个方案。

上一章介绍了伪代码的概念，把它作为表示算法的一种方法。这一章将开始分析要把伪代码翻译成的程序设计语言。就像每把锁都有专用的钥匙一样，每种类型的计算机都有一套它能够执行的专用操作，称为计算机的机器语言。我们从机器代码着手讨论程序设计语言。由于我们从不凭空编写程序，所以在这一章中既有语言表示法，又有对应的伪代码。

目标

学完本章之后，你应该能够：

- 列出计算机能够执行的操作。
- 讨论抽象分层和具体算法步骤的确定之间的关系。
- 描述虚拟机Pep/7的重要特征。
- 区分立即方式寻址和直接寻址技术。
- 把简单的算法翻译成机器语言程序。
- 区分机器语言和汇编语言。
- 把简单的算法翻译成汇编语言程序。
- 区分给汇编器的指令和要翻译的指令。
- 为简单的汇编语言程序设计和实现测试方案。

7.1 计算机操作

无论是表示算法的所有符号，还是用于实现算法的程序设计语言，都必须反映出计算机能够执行的操作类型。让我们通过重述计算机的定义来开始新的讨论：计算机是能够存储、检索和处理数据的可编程电子设备。

这个定义中的操作字包括*可编程的*、*存储*、*检索*和*处理*。上一章指出了数据和操作数据的指令逻辑上是相同的，它们存储在相同的地方。操作数据的指令就存储在数据所在的机器上。要改变计算机对数据的处理，只需要改变指令即可。改变指令的能力就是“可编程的”这个词的意义所在。

存储、检索和处理是计算机能够对数据执行的动作。也就是说，控制器执行的指令能够把数据存储到机器的内存中，在机器内存中检索数据，在算术逻辑部件中以某种方式处理数据。词语“处理”非常通用。在机器层，处理涉及在数据值上执行算术和逻辑操作。

那么，要存储到计算机内存中的数据来自何处呢？人们如何查看内存中存储的是什么（例如查看某个计算的结果）？规定输入设备与CPU之间以及CPU与输出设备之间的交互的是

另外一些指令。

7.2 抽象的分层

在第6章描述问题求解的过程时，我们说过，抽象步骤是某些细节还未明确的步骤，具体步骤是细节完全明确的步骤。那么，如何知道一个步骤何时才算是具体的？答案是由用来表示算法的程序设计语言决定的。

在第6章的地址列表实例中，我们假设步骤“对列表排序”已经完全明确了，而步骤“输出列表”还未明确。在某些程序设计语言中，这些假设是成立的。但是，在另外一些程序设计语言中，“对列表排序”这个步骤是抽象的，“输出列表”是具体的。此外，还有一些程序设计语言，其中这两个步骤都是抽象的。

这一章和下一章要详细地介绍如何用伪代码编写算法，以及如何把伪代码翻译成程序设计语言。首先介绍的是硬件附带的语言——*机器语言*，然后介绍低级程序设计语言——*汇编语言*，最后是第8章介绍的*高级程序设计语言*。每前进一个阶段，语言自身就变得越抽象，也就是说，用语言中的一个语句可以表达的处理会更复杂。你也许已经想到了，这种从具体到抽象的演化过程反映的正是软件开发的历史。

7.3 机器语言

第1章提到过，计算机真正执行的程序设计指令是用机器语言编写的指令，这些指令固定在计算机的硬件中。起初，人们只能用机器语言编写指令，因为当时还没有发明其他程序设计语言。

> **机器语言**（machine language）：由计算机直接使用的二进制编码指令构成的语言。

那么计算机指令是如何表示的呢？每种处理器都有自己专用的机器指令集合。这些指令是处理器唯一真正能够执行的指令。由于指令的数量有限，所以处理器的设计者就列出所有的指令，给每个指令分配一个二进制代码，用来表示它们。这与第3章中介绍的表示字符数据的方法相似。

处理器与它能够执行的指令之间的关系十分和谐。CPU的电子器件本来就能够识别专用命令的二进制表示，因此，计算机必须参考的命令的真实清单并不存在。CPU把这个清单嵌入了自己的设计。

每条机器语言指令只能执行一个非常低级的任务。在机器语言中，处理过程中每一个微小的步骤都必须明确地编码。即使是求两个数的和这样的小任务，也需要三条用二进制编写的指令。程序员必须记住每组二进制数对应的是什么指令。如第1章所述，机器语言的程序员必须对数字很敏感，而且非常注意细节。

但是，并非只有数学家才能用机器语言编写程序，我们不想给你留下这样的印象。事实上，目前几乎没有程序是用机器语言编写的，主要是因为编写这种程序太费时间。大多数程序是用高级语言编写，然后翻译成机器语言的，我们将在下一章介绍这一过程。但是，每个人都应该体验一下早期的先驱们用特定机器的机器代码编写第一个程序的感受。这种体验会强化计算机的基本定义，使你感谢今天能够如此容易地与计算机进行交流。

Pep/7：一台虚拟计算机

根据机器代码的定义可知，不同机器的机器代码不同。也就是说，每种类型的CPU都有自己能够理解的机器语言。因此，对于使用不同机器的读者，如何才能让你们体验用机器语言编码呢？我们使用一台虚拟计算机解决这个问题。所谓**虚拟计算机**，就是一台假想的机器，这台机器具有我们想说明的真实计算机的重要特性。我们此处使用的虚拟机是Stanley Warford设计的Pep/7。[1]

> **虚拟计算机**（virtual computer）：用于说明真实计算机的重要特性的假想机。

Pep/7有32条机器语言指令。这意味着Pep/7的程序都是由这32条指令的组合构成的序列。不必惊慌，我们不会要求你理解并记住这32条二进制数列。我们只想分析几条指令，不要求你记住它们中的任何一条。

Pep/7反映的重要特性

Pep/7的内存单元由4096个字节的存储器构成，字节的编号是十进制数0到4095。由于每个字节包含8个位，所以可以用2个十六进制数字描述一个字节的位组合（关于十六进制数字的信息请参阅第2章）。Pep/7的字长是2字节或16位。因此，流入和流出ALU（算术逻辑部件）的信息长度是16位。

第5章提到过，寄存器是CPU的算术逻辑部件中的一小块存储区，用于存放特殊数据和中间值。Pep/7有7个寄存器，这里我们着重介绍其中的3个，另外还有4个状态位，我们将分析其中的两个。这些寄存器和状态位如下：

- 程序计数器（PC），存放的是下一条要执行的指令的地址。
- 指令寄存器（IR），存放的是正在执行的指令的副本。
- 累加器（寄存器A）。
- 状态位N，如果寄存器A是负数，该位为1，否则为0。
- 状态位Z，如果寄存器A是0，该位为1，否则为0。

累加器用于存放操作的数据和结果，是一种特殊的存储寄存器，请参阅第5章对ALU的讨论。状态位给出了关于寄存器A（累加器）的内容的信息。

我们知道，这些信息错综复杂，不过不要绝望。记住，我们的目标是让你了解在最底层的计算机处理中会发生些什么，这不可避免地会涉及许多细节。

图7-1是Pep/7的CPU和内存的图解。注意，内存中的地址本身并不存储在内存中，它们只是“命名”了内存中的独立字节。我们用地址引用内存中的特定字节。

在讨论下一个论题前，让我们先回顾一下二进制数和十六进制数。一个字节能够表示的最大十进制数是255，用二进制表示是11111111，用十六进制表示是FF。一个字（16位）能够表示的最大十进制数是65 535，用二进制表示是1111111111111111，用十六进制表示是FFFF。如果既要表示正数，又要表示负数，那么在量级上就会少一位（因为有一位用于表示符号），因此可以表示的十六进制值的范围约为－7FFF到+7FFF，相当于十进制数的－32 767到＋32 767。

在使用Pep/7时，这些信息非常重要。我们能用的位数决定了可以使用的数的大小。

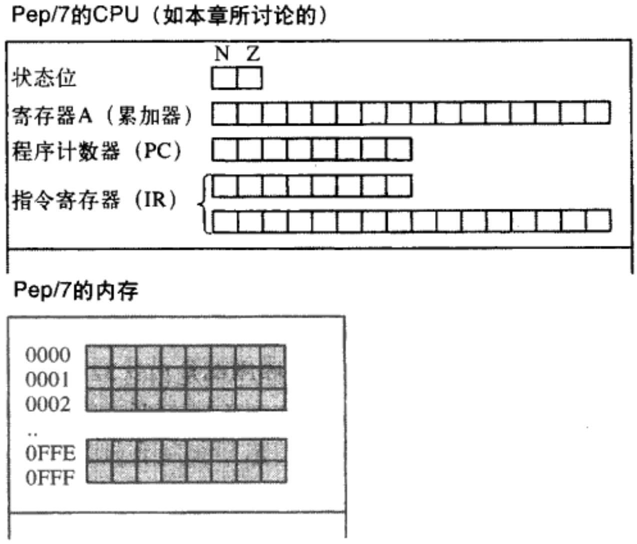


图7-1　Pep/7的体系结构

指令格式

我们说过，指令要先进入指令寄存器，然后经过译解，最后被执行。接下来，让我们仔细研究一套计算机能够执行的具体指令。首先，我们需要分析Pep/7中的指令格式。

图7-2展示了Pep/7中的指令格式。一条指令由两部分组成，即8位的*指令说明符*和（可选的）16位的*操作数说明符*。指令说明符（指令的第一个字节）说明了要执行什么操作（如把一个数加到一个已经存储在寄存器中的值上）和如何解释操作数的位置。操作数说明符（指令的第二和第三个字节）存放的是操作数本身或者操作数的地址。有些指令没有操作数说明符。（指令说明符和操作数说明符通常简称为操作和运算操作数。）

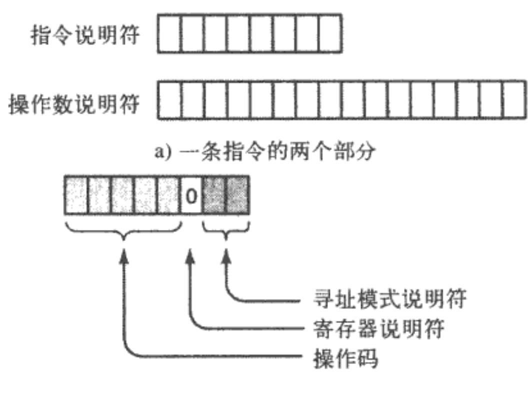


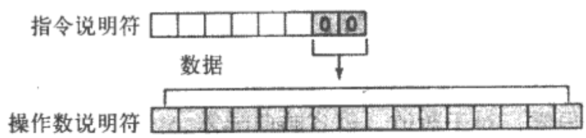
图7-2　Pep/7的指令格式

指令说明符由几部分构成，即操作码、寄存器说明符和寻址模式说明符。*操作码*是5位长的位串，说明了要执行哪条指令。当我们介绍Pep/7具有32条指令时，也许你已经预见到操作码的长度是5位了（5个位可以表示32种不同的组合）。

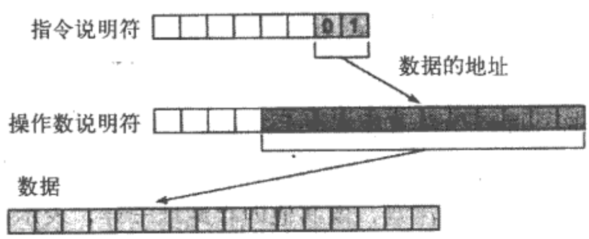
1位的*寄存器说明符*是0，因为寄存器A（累加器）是唯一要用到的寄存器。寄存器说明符

存放的总是0。

2位的*寻址模式说明符*说明了如何解释指令的操作数部分。如果寻址模式是00，将操作数看成是16位整数。也就是说，操作数是数据，而不是数据的地址。这种寻址模式称为立即寻址 (i)。如果寻址模式是01，操作数存放在操作数说明符指定的内存地址处。这种寻址模式称为直接寻址 (d)。（此外还有两种寻址模式，这里没有介绍。）立即寻址模式和直接寻址模式的区别非常重要，因为它们决定了操作数使用的数据存放或要存放的位置。如图7-3所示。存放地址的区域以深灰色显示，操作数部分以浅灰色显示。



a) 立即寻址模式，操作数涂成了灰色



b) 直接寻址模式，操作数涂成了灰色

图7-3 立即寻址模式和直接寻址模式的区别

没有操作数（要处理的数据）的指令称为*一元指令*，没有操作数说明符。也就是说，一元指令的长度是1个字节，而不是3个字节。

一些示例指令

让我们逐条研究一些指令，然后把它们组合在一起，编写一个程序。图7-4中是7种操作的5位操作码。回忆一下，操作码是指令说明符最左边的5位，第6位说明的是使用的寄存器（如果使用了），余下的2位是寻址模式说明符。

操作码	指令的含义	操作码	指令的含义
00000	停止执行	00100	从寄存器A中减去操作数
00001	把操作数载入寄存器A	11011	把字符输入操作数
00010	把寄存器A的内容存储到操作数中	11100	从操作数输出字符
00011	把操作数加到寄存器A中		

图7-4 Pep/7的部分指令

00000*停止执行* 在读取–执行周期中，当操作码全部为0时，程序将停止。停止指令是个一元指令，所以它只占用一个字节，而且这个字节中最右边的三位将被忽略。

00001*把操作数载入寄存器A* 这条指令将把一个字（两个字节）载入寄存器A。模式说明符决定了这个字要载入的位置。因此，寻址模式说明符不同，载入操作码的含义就不同。模式说明符决定了要载入的值是存放在指令的操作数部分（指令的第二个字节和第三个字节）

还是存放在操作数指定的位置。让我们看看这两种情况的具体例子。下面是一条指令的前三个字节。

指令说明符 00001000
操作数说明符 0000000000000111

寻址模式是立即寻址，即要载入寄存器A的值存放在操作数说明符中。也就是说，数据就存放在操作数说明符中，因此，操作数说明符被涂成了浅灰色。执行这条指令后，指令的第二个字节和第三个字节（操作数说明符）中的内容将被载入寄存器A（累加器）。也就是说，寄存器A存放的将是0007，原来的内容将丢失。

为了方便讨论，从此我们将用十六进制表示所有数字（除了寄存器中的位串）。只有引用地址时，我们才说明前导零，对于数值，则省略前导零。

下面是另一条载入指令。

指令说明符 00001001
操作数说明符 0000000000011111

寻址模式是直接寻址，即操作数本身不存在于操作数说明符（指令的第二和第三个字节）中，操作数说明符中存放的是操作数所在的内存的地址。因此，在执行这条指令时，存储单元001F的内容将被载入寄存器A。注意，像其他内存地址一样，表示内存地址的位被涂成了深灰色。寄存器A存放的是一个字（2字节），因此当像这个例子中所示的，地址指定的是一个字（而不是一个字节）时，给出的地址是一个字左边的字节。因此，相邻的两个内存单元001F和0020中的内容将被载入寄存器A。操作数（001F和0020）的内容并未改变。

00010**把寄存器A中的内容存储到操作数中** 这条指令将把寄存器A中的内容存储到操作数指定的位置，既可以是操作数本身，也可以是操作数指定的位置。

指令说明符 00010001
操作数说明符 0000000000001010

这条指令将把寄存器A中的内容存储到从内存单元000A开始的字中。在存储操作码中，立即寻址模式是无效的，也就是说，不能把存储器中的内容存储到操作数说明符中。

00011**把操作数加到寄存器A中** 与载入操作相似，加法操作使用寻址模式说明符，对操作数做出不同的解释。下面列出了这条指令的两个例子，每个例子后有相应的说明。

指令说明符 00011000
操作数说明符 0000001000001010

这条指令的第二和第三个字节（操作数说明符）中的内容（20A）将被加到寄存器A中的内容上。因此，操作数说明符被涂成了浅灰色，以说明它是数据。

指令说明符 00011001
操作数说明符 0000[illegible]

这条指令的第二和第三个字节中说明的操作数的内容（地址020A）将被加到寄存器A中。

00100从寄存器A中减去操作数 这条指令与加法操作的指令相似，只不过是从寄存器A中减去操作数，而不是把操作数加到寄存器A中。与载入操作和加法操作一样，寻址模式决定了这条指令的变体。

11011把字符输入操作数 这条指令允许程序在运行时从输入设备输入一个ASCII字符。由于它只能使用直接寻址模式，所以输入的字符将存储在操作数说明符指定的地址中。

指令说明符	1 1 0 1 1 0 0 1
操作数说明符	0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0

这条指令将从输入设备读入一个ASCII字符，并把它存储在内存单元000A中。

11100从操作数输出字符 这条指令将在程序运行时把一个ASCII字符发送到输出设备。寻址模式可以是立即寻址，也可以是直接寻址。

指令说明符	1 1 1 0 0 0 0 0
操作数说明符	0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1

因为这条指令采用的是立即寻址模式，所以它输出的是存储在操作数说明符中的ASCII字符。操作数说明符存放的是1000001，即十六进制的41或者十进制的65，对应的ASCII字符是“A”，因此屏幕上将显示字符A。

指令说明符	1 1 1 0 0 0 0 1
操作数说明符	0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0

因为采用的是直接寻址模式，所以这条指令输出的是存储在操作数说明符指定的内存单元000A中的ASCII字符。输出的究竟是什么字符呢？除非我们知道000A中存放的内容，否则不能确定。无论存储在这个内存单元中的内容对应的是哪个ASCII字符，它都将被输出。

7.4 一个程序实例

让我们分析一个用机器语言编写的程序实例。程序是用来解决问题的，所以首先我们将提出一个问题，并设计它的解决算法。

7.4.1 问题和算法

我们从一个非常简单的问题着手，即在屏幕上显示“Hello”。算法非常简单。

Write “Hello”

这是一个具体步骤吗？如果用高级程序设计语言实现这个步骤，它当然是一个具体步骤。但是，在机器语言中，它却不是具体步骤。我们把它进一步分解为单独显示每个字母。

<u>Write “Hello”</u>

Write “H”
Write “e”
Write “l”
Write “l”
Write “o”

这些是具体步骤吗？在机器语言中，它们仍然不是具体步骤，必须把字母转换成它们的ASCII表示法。

Write "H"

Write 48 (hex)

Write "e"

Write 65 (hex)

Write "l"

Write 6C (hex)

Write "l"

Write 6C (hex)

Write "o"

Write 6F (hex)

现在，每个步骤都是具体步骤了。

7.4.2 程序

现在，我们已经准备好用机器语言实现在屏幕上显示“Hello”的这个算法了。这个程序有6条指令，其中5条用于显示字符，另外一条用于指示过程结束了。在屏幕上显示字符的指令是11100，即“从操作数输出字符”的操作。那么应该把字符存储在内存中，使用直接寻址模式输出它们呢，还是应该把字符存储在操作数说明符中，使用立即寻址模式呢。这里使用立即寻址模式，把直接寻址模式留作练习。这意味着寻址模式说明符是00，ASCII码将存放在指令的第三个字节中。

模　块	二进制指令	十六进制指令
Write "H"	11100000	E0
	0000000001001000	0048
Write "e"	11100000	E0
	0000000001100101	0065
Write "l"	11100000	E0
	0000000001101100	006C
Write "l"	11100000	E0
	0000000001101100	006C
Write "o"	11100000	E0
	0000000001101111	006F
Stop	00000000	00

这个表的第二列展示的是用二进制表示的机器语言程序，第三列展示的是用十六进制表示的程序。注意要用二进制构造操作数说明符，因为它由5位操作码、1位寄存器说明符和2位寻址模式说明符构成。一旦凑够了8位，就可以把它转换成十六进制的。可以直接用十六进制构造操作数说明符。

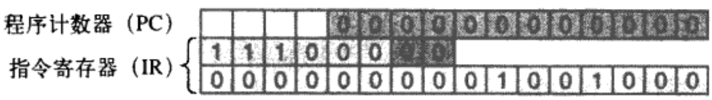
手工模拟

让我们通过执行读取-执行周期的步骤，模拟这个程序的执行。这种手动跟踪的方式绝对能让你理解计算机执行的步骤。

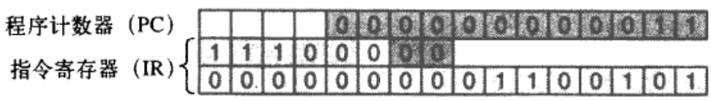
回忆一下读取–执行周期的四个步骤：

- （从程序计数器指定的位置）读取下一条指令
- 译解指令（并且更新程序计数器）
- 如果需要，获取数据（操作数）
- 执行指令

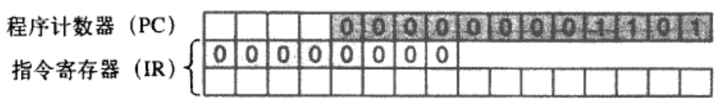
在我们的程序中有6条指令。假设它们存放在内存的连续单元中，第一条指令存储在内存的0000–0002单元中。执行的第一步是把0000载入程序计数器（PC）。我们将分析执行过程中每个阶段内的PC（程序计数器）和IR（指令寄存器）。由于这个程序不访问寄存器A，所以这里不再分析它。在第一个读取操作结束，PC还没有改变前，PC和IR如下图所示。



这条指令将被译解为用立即寻址模式把字符输出到输出设备。因为这条指令有三个字节，所以PC将加3。从操作数说明符获得的数据将被存储到IR中，该指令将被执行。字母“H”将出现在屏幕上。第二个读取操作执行后，PC和IR如下图所示。



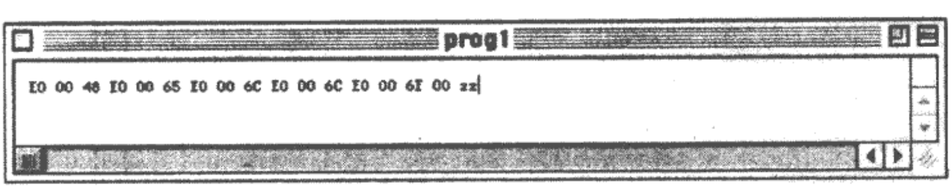
这条指令被译解为用立即寻址模式把字符输出到输出设备。因为这条指令有三个字节，所以PC将加3。从操作数说明符获得的数据将被存储到IR中，该指令将被执行，字母“e”将出现在屏幕上。执行随后三条指令的方式完全相同。当“o”出现在屏幕上后，PC和IR如下图所示。

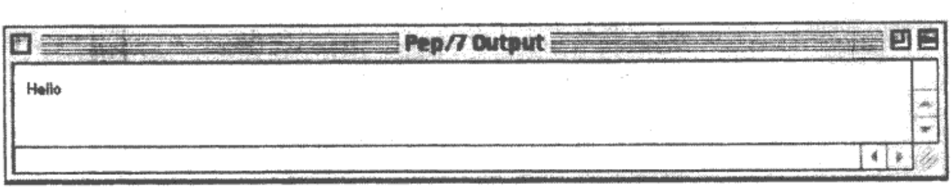


这条操作码被译解为停止指令。寻址模式和操作数说明符的内容将被忽略。读取–执行周期将停止。

Pep/7模拟程序

记住，这些指令是用Pep/7的机器语言编写的，这种语言不反映任何特定CPU的机器语言。我们已经手动模拟了这个程序，那么可以在计算机上执行它吗？可以。虽然Pep/7是一台虚拟机，但是它有对应的模拟程序。也就是说，我们有一个行为与Pep/7虚拟机的行为一





样的程序。要运行一个程序，需要输入十六进制的代码，每个字节之间用空格隔开，以zz结束程序。模拟程序可以识别程序结尾处的两个z。下面是Pep/7机器语言程序的屏幕图和输出窗口的屏幕图。



让我们执行一次从算法转换到输出所必需的步骤。假设已经安装了Pep/7虚拟程序。使用菜单条上的File菜单，打开一个新文件。

然后输入上面的图中所示的程序，保存该文件，给它起一个名字。

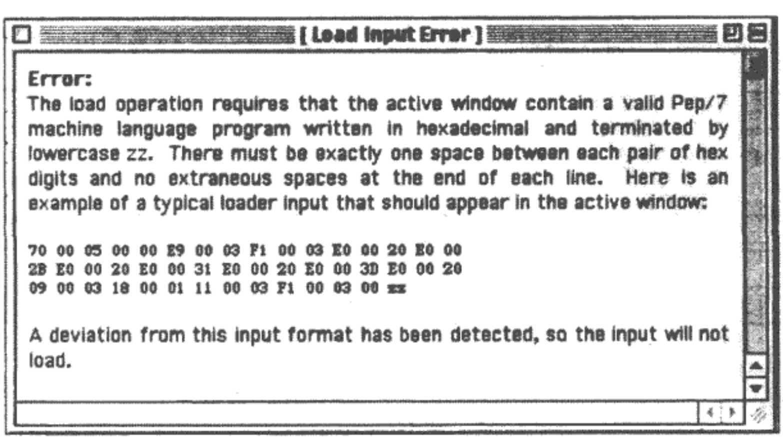
下一步是把程序载入Pep/7虚拟机的内存，并执行它。在Pep/7的下拉式菜单中有几个选项。第二个选项是Loader，点击这个选项后，该软件中的**装入程序**将读取我们的程序，把它载入从0000存储单元开始的内存。下表是装入程序运行完后Pep/7内存的情形。虽然内存是一个很长的字节流，但这里用三个字节一组的形式表示它，因为除了停止指令外，其他指令都是三个字节的。

地址			
00	E0	00	48
03	E0	00	65
06	E0	00	6C
09	E0	00	6C
0C	E0	00	6F
0F	00		

> **装入程序**（loader）：软件用于读取机器语言程序并把它载入内存的部分。

读取－执行周期从0000内存单元中的指令开始，这里存储的是要载入的第一条指令。要启动读取－执行周期，需要返回Pep/7下拉式菜单，点击其中的Execute选项。若要把载入和执行操作合并到一步，点击Pep/7菜单中的Load/Execute选项即可。

把程序载入内存的装入程序是非常严格的。指令必须是用十六进制编写的，每个字节之间只能有一个空格。如果输错了程序，例如在zz前忘记了输入空格，那么装入程序将给出下列消息：





John von Neumann[2]

John von Neumann是著名的数学家、物理学家、逻辑学家和计算机科学家。他那令人吃惊的记忆力和解决问题的神速，给他涂上了一层传奇色彩。他不仅能用自己的天分推动数学理论的研究，还能记住整本的书，过了一年之后再把它们复述出来。不过，如果问一个高速公路的巡警von Neumann的驾驶技术如何，他一定会无奈地耸耸肩，这位数学天才坐在方向盘面前就会像十几岁的少年一样反叛，不计后果。

John von Neumann 1903年生于匈牙利，是一个富有的犹太银行家的长子。6岁时，他就能够心算8位数字的除法。11岁时，他进入了中学，没过多久，他的数学老师就推荐了一位大学教授指导他。他的父亲一直希望他学习一些赚钱的方法，作为妥协，他于1921年在柏林大学注册，开始学习化学。1926年，他在苏黎世的Technische Hochschule得到了化学工程学的毕业证书。同一年，他获得了布达佩斯大学授予的数学博士学位，毕业论文是关于集合论的。在1926年到1929年这段时间中，von Neumann在柏林执教，同时得到了Rockefeller奖学金，在汉堡的哥廷根大学从事博士后研究。

20世纪30年代早期，von Neumann移居到了美国，在普林斯顿执教，同时保有德国的学术职位。尽管他不像许多人一样是政治流亡者，但当纳粹当政时，他辞去了德国的职位。在普林斯顿的这段时间，他与当时还济济无名的英国学生Alan Turing合作过。他继续致力于自己辉煌的数学生涯，成为了《Annals of Mathematics》的编辑和《Compositio Mathematica》的合编者。在世界大战期间，由于von Neumann具备流体力学的知识，所以他受雇于美国军方和相关的非军界机构，担任顾问。此外，1943年他还被召集参加了原子弹研发工作。经过这些工作后，美国总统Eisenhower于1955年任命他为美国原子能委员会的一员。

尽管原子弹和它们的性能强烈地吸引住了von Neumann很多年，但是1944年与Herbert Goldstine（第一台可操作的电子数字计算机的发明者之一）的一次偶然相遇，却使这位数学家知道了比原子弹更重要的东西——计算机。von Neumann在一个火车站与Goldstine的偶然交谈燃起了他对新事物的兴趣之火。他开始致力于存储程序这个概念的研究，并得出结论，内部存储一个程序能够减少为计算机重新编程所需要的繁重工作。他还开发了一种新的体系结构来执行这种存储任务。事实上，当今的计算机通常被称为冯·诺伊曼机，因为他描述的体系结构被证明为极其成功。在过去40年中，计算机的变化主要是速度和基础电路的构成法，而von Neumann设计的基本体系结构却保持未变。

在20世纪50年代，von Neumann担任IBM的顾问，在此他检阅了许多刚提出的或正在实施的高科技项目。John Backus的FORTRAN就是这些项目中的一个，传说von Neumann对这个项目进行了质疑，问为什么有了一种机器语言，人们还需要其他机器语言。1957年，von Neumann在华盛顿死于骨癌，享年54岁。也许是他曾经从事的原子弹工作导致了骨癌，致使20世纪最具奇思妙想的伟人之一离我们而去了。

7.5 汇编语言

第1章提到过，开发的第一种帮助程序员的工具就是汇编语言。**汇编语言**给每条机器语言

指令分配了一个助记忆指令码，程序员可以用这些指令码代替二进制数字。汇编语言中的指令与手持计算器的按钮上显示的指示相似。

因为在计算机上执行的每个程序最终都要被翻译成机器语言的形式，所以一个名为**汇编器**的程序将读取每条指令的助记忆码，然后把它翻译成等价的机器语言。因为每种类型的计算机都有自己的机器语言，所以有多少种机器，就有多少种汇编语言和翻译程序。

> **汇编语言**（assembly language）：一种低级语言，用助记忆码表示特定计算机的机器语言指令。
> **汇编器**（assembler）：把汇编语言程序翻译成机器代码的程序。

> **音乐智能**
>
> Platinum Blue开发了一个软件，用于预测哪些歌将流行。该软件使用了“光谱解卷积”（spectral deconvolution）法分析每首歌的特征，如“弯曲度”（开头有各种音调，结尾则只有一种音符）、节奏、连续的和弦和音质，报告的成功率达80%。该软件还能发现不同时期的、看似无关的两首歌中的相似点。Platinum Blue的这个软件显示，U2乐队的某些歌曲与贝多芬的某些音乐有明显的相似之处。

7.5.1 Pep/7汇编语言

这一节的目标不是要使你成为汇编程序员，而是让你了解使用汇编语言进行程序设计比使用机器代码的好处。有了这个目标，我们就只介绍Pep/7汇编语言的几点特性，从分析上一节中的操作着手。在Pep/7汇编语言中，每个寄存器有一个操作码，操作数是十六进制的，由“h#”说明，寻址模式说明符由字母i或d说明。

助记忆码	操作数、寻址模式说明符	指令的含义
STOP		停止执行
LOADA	h#008B, i	把008B载入寄存器A
LOADA	h#008B, d	把内存单元8B中的内容载入寄存器A
STOREA	h#008B, d	把寄存器A中的内容存入内存单元8B
ADDA	h#008B, i	把008B加到寄存器A中
ADDA	h#008B, d	把内存单元8B中的内容加到寄存器A中
SUBA	h#008B, i	从寄存器A中减去008B
SUBA	h#008B, d	从寄存器A中减去内存单元8B中的内容
CHARI	h#008B, d	读取一个字符，把它存入内存单元8B中
CHARO	c#/B/, i	输出字符B
	h#008B, d	输出存储在内存单元8B中的字符
DECI	h#008B, d	读取一个十进制数，把它存储在内存单元8B中
DECO	h#008B, i	输出十进制数139（即十六进制的8B）
DECO	h#008B, d	输出存储在内存单元8B中的十进制数

7.5.2 伪代码操作

在机器语言程序中，每条指令都要先存储到内存中，然后才能执行。从汇编语言开始，大多数程序设计语言都有两种类型的指令，即要翻译的指令和翻译程序使用的指令。下面是Pep/7汇编器中的几条有用指示（即给汇编器的指令）。

伪代码操作码	操作数	含义
.ASCII	/.../	把//之间的字符存储到内存中
.BLOCK	d#3	生成三个字节的存储空间，并把每个字节设置为0
.WORD	d#5	生成一个字的存储空间，并把十进制值5存储进去
.WORD	h#0105	生成一个字的存储空间，并把十六进制值0105存储进去
.END		表示汇编语言程序的终点

7.5.3 “Hello”程序的汇编语言版本

让我们再看看这个算法，即在屏幕上显示“Hello”。

Write “Hello”

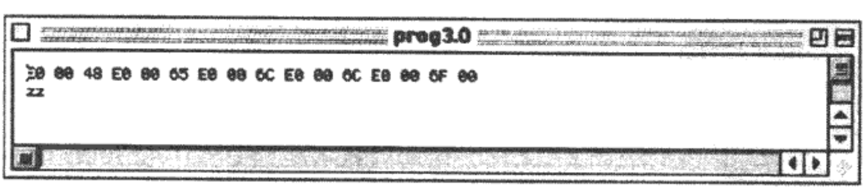
Write “H”
Write “e”
Write “l”
Write “l”
Write “o”

对于机器语言程序，必须进一步明确这个模块中的每一个步骤。由于汇编语言能够直接指定要输出的字符，所以如果使用汇编语言，而不是机器语言，那么这个模块就是具体的，可以对模块中的每个步骤直接编码。除了指令外，汇编语言还允许我们添加**注释**。注释是为程序的使用者而写的说明性文字，解释了会发生什么情况。无论编写什么程序，注释都是一个重要部分。汇编器会忽略从分号开始到一行结束处的所有字符。

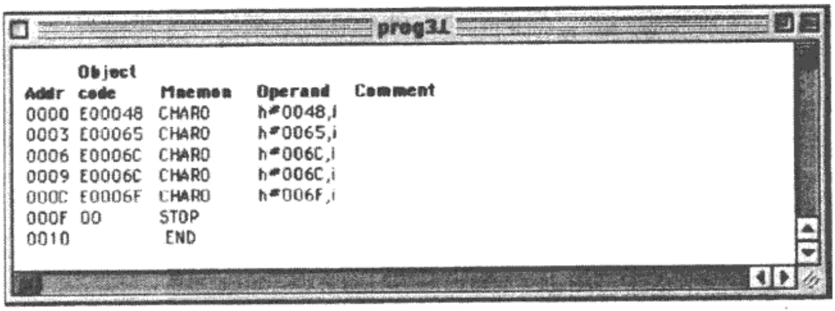
> **注释**（comment）：为使用者而写的说明性文字。

```
CHARO C#/H/,i   ;Output 'H'
CHARO C#/e/,i   ;Output 'e'
CHARO C#/l/,i   ;Output 'l'
CHARO C#/l/,i   ;Output 'l'
CHARO C#/o/,i   ;Output 'o'
STOP
.END
```

运行Pep7菜单中的Assemble选项，结果如下图所示。



与7.4.2节中的机器语言程序图进行比较，它们是完全相同的。汇编器输出的就是这个程序的机器语言版本。有了机器语言的程序后，就可以按照执行机器语言的版本所用的方法执行它。此外，点击Pep7菜单中的Assembler Listing选项，还可以得到一个汇编器清单，如下图所示。



prog31

Addr	Object code	Mnemon	Operand	Comment
0000	E00048	CHARO	h#0048,i	
0003	E00065	CHARO	h#0065,i	
0006	E0006C	CHARO	h#006C,i	
0009	E0006C	CHARO	h#006C,i	
000C	E0006F	CHARO	h#006F,i	
000F	00	STOP		
0010		END		

图7-5说明了这一过程。输入汇编器的是用汇编语言编写的程序。从汇编器输出的是用机器代码编写的程序。由此你会明白，为什么汇编语言的出现是程序设计语言发展史上的重要一步。它把指令抽象成几个单词，从而消除了机器语言程序设计中的繁琐细节。尽管它在程序执行过程中加了一个步骤，但这个额外的步骤却大大简化了程序员的工作。



图7-5　汇编过程

7.5.4　一个新程序

让我们编写一个复杂些的程序，读入三个数字，然后输出它们的和。我们如何手动地完成这个任务呢？如果有一个计算器，我们首先会把总值清零，也就是把和设置为0。然后输入第一个数，把它加到总值上，再输入第二个数，把它加到总值上，最后输入第三个数，把它加到总值上。结果将存储在计算器的累加器上。我们可以根据这种算法对程序建模。

```
Reading and adding three numbers

Set sum to 0
Read num1
Add num1 to sum
Read num2
Add num2 to sum
Read num3
Add num3 to sum
Write sum
```

第一步是具体步骤，它把一个内存单元设置为0。事实上，所有语句看来都是具体的。最复杂的是，这里有四个标识符，必须把它们和内存单元关联起来，也就是说，如果把数据放在程序之后，就要知道程序自身需占用多少个内存单元。我们把数据放在程序之前，从而简化了这个过程。我们把标识符关联到从0001开始的内存单元，让读取-执行周期在运行程序时跳过这些内存单元。实际上，可以给内存单元分配标识符，然后在程序中使用这些名字即

可。我们用伪代码.WORD为总值创建了空间，这样就可以把和设置为0，然后用伪代码.BLOCK为三个数字创建空间。

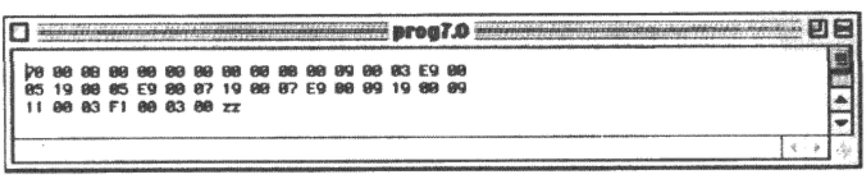
```
sum:  .WORD d#0      ;set up word with zero as the contents
num1: .BLOCK d#2     ;set up a two byte block for num1
num2: .BLOCK d#2     ;set up a two byte block for num2
num3: .BLOCK d#2     ;set up a two byte block for num3
```

我们可以引用这些标识符，汇编器用它们代替了地址。现在，所有步骤都是具体的了。算法中的第一步是把和设置为0。伪代码.WORD已经实现这一步了。接下来的两步将被重复三次，即读入一个数字，把它加到和上。糟糕，算法中缺少一个重要步骤，没有把累加器清零。我们首先要把和载入累加器（以把它设置为0），或者把第一个数载入累加器，而不是把这个数加到累加器上。由于我们确定和是0，所以先把它载入。在读入三个数字，并把它们加到和上之后，必须输出和。糟糕，这一步也漏掉了。我们必须保存累加器的值，以便输出。

下面是完整的程序。注意，第一个语句是分支语句，以绕过其后的数值。也就是说，执行将从存储在内存单元0000中的指令开始。由于我们把数据放在程序的开头，所以在0000这个内存单元中就要存放一条指令，以便把程序中第一条指令的地址放入程序计数器中。BR指令执行的正是这一操作，BR Main这个语句将把与标识符Main关联在一起的内存地址存入程序计数器，以便存放在Main中的指令能被载入指令寄存器并执行。我们排列了这些语句，以便使它们更容易阅读。

```
      BR   Main     ;branch to location Main
sum:  .WORD d#0     ;set up word with zero as the contents
num1: .BLOCK d#2    ;set up a two byte block for num1
num2: .BLOCK d#2    ;set up a two byte block for num2
num3: .BLOCK d#2    ;set up a two byte block for num3
Main: LOADA sum,d   ;load a copy of sum into accumulator
      DECI num1,d   ;read and store a decimal number in num1
      ADDA num1,d   ;add the contents of num1 to accumulator
      DECI num2,d   ;read and store a decimal number in num2
      ADDA num2,d   ;add the contents of num2 to accumulator
      DECI num3,d   ;read and store a decimal number in num3
      ADDA num3,d   ;add the contents of num3 to accumulator
      STOREA sum,d  ;store contents of the accumulator into sum
      DECO sum,d    ;output the contents of sum
      STOP          ;stop the processing
      .END          ;end of the program
```

下面是这个程序的机器代码和汇编器清单。仔细研究这两个图，确保你理解了程序是如何运行的。



```
PROG9.L

      Object
Addr  code    Symbol   Mnemon   Operand   Comment
0000  70000B           BR       Main      ;branch to location Main
0003  0000    sum:     .WORD    d#0       ;set up word with zero as the contents
0005  0000    num1:    .BLOCK   d#2       ;set up a two byte block for num1
0007  0000    num2:    .BLOCK   d#2       ;set up a two byte block for num2
0009  0000    num3:    .BLOCK   d#2       ;set up a two byte block for num3
000B  090003  Main:    LOADA    sum,d     ;load a copy of sum into accumulator
000E  E90005           DECI     num1,d    ;read and store a decimal number in num1
0011  190005           ADDA     num1,d    ;add the contents of num1 to accumulator
0014  E90007           DECI     num2,d    ;read and store a decimal number in num2
0017  190007           ADDA     num2,d    ;add the contents of num2 to accumulator
001A  E90009           DECI     num3,d    ;read and store a decimal number in num3
001D  190009           ADDA     num3,d    ;add the contents of num3 to accumulator
0020  110003           STOREA   sum,d     ;store contents of the accumulator into sum
0023  F10003           DECO     sum,d     ;output the contents of sum
0026  00               STOP               ;stop the processing
0027                   .END               ;end of the program

Symbol      Value        Symbol      Value
Main        000B         num1        0005
num2        0007         num3        0009
sum         0003
```

7.5.5 具有分支的程序

如果每次只能对程序计数器加1，那么程序的用途就不大了。从第6章可以知道，我们能够用算法提问。那么可以用Pep/7提问吗？当然可以，只是不像用伪代码提问那么简单。还记得吗，Pep/7有四个状态位，在7.3.1节中我们介绍了其中的两个——N和Z。可以用操作码查询这些状态位的状态。下面是两个有用的操作码和它们的含义。

助 记 忆 码	操作数、寻址模式说明符	指令的含义
BRLT	无	如果N是1（寄存器A是负数），则把PC设置为操作数
BREQ	无	如果Z是1（寄存器A是零），则把PC设置为操作数
COMPA	H#008B, i	对比累加器与地址008B中的内容，设置状态位
COMPA	H#008B, d	对比累加器与它的操作数中的内容，设置状态位

当累加器为负数时，状态位N是1。当累加器为0时，状态位Z是1。如何设置这些状态位呢？状态位是由前面执行的语句决定的。例如，

```
LOADA num1, d
BRLT lessThan     ;Branch to lessThan if num1 is less than 0
```

在把num1载入累加器时，如果它的值是负数，那么状态位N将被设置为1。BRLT将测试状态位N的值，如果是1，则把程序计数器设置为lessThan的内存地址。如果状态位N不是1，那么PC保持不变。

让我们更改一下前面的程序，如果三个数的和是正数，则输出和，如果和是负数，则输出错误消息。在哪里进行测试呢？就在要把计算出的结果存入地址sum之前，可以测试寄存器A，如果它是负数，则输出“Error”。在重新编码之前，我们先用伪代码编写这些指令。

```
...
Add num3 to sum
If sum is negative
```

```
    Write "Error"
else
    Write sum
```

可以用BRLT指令测试和是否为负数。如果状态位N是1，那么PC的内容将被BRLT后面的操作数代替，使下一条指令从操作数指定的内存单元开始。必须给这条指令一个名字，我们称之为NegMsg。在伪代码中，可以用“Go to NegMsg”。当显示出错误消息后，必须跳转到让程序结束的STOP行，也就是说，必须给这个代码行一个名字。我们称之为Quit。注意，在汇编语言中，会给存放下一条要执行的指令的内存地址一个名字。下面是以算法形式扩展的If语句。

```
...
Add num3 to sum
If status bit N is 1
    Go to NegMsg
Write sum
Quit: STOP
NegMsg: Write the message and go to Quit
```

下面是包含程序的汇编清单。仔细阅读这个清单，确保你理解了分支语句的控制流。

```
prog11

     Object
Addr code    Symbol   Mnemon  Operand   Comment

0000 70000B           BR      Main      ;branch to location Main
0003 0000    sum:     .WORD   d#0       ;set up word with zero as the contents
0005 0000    num1:    .BLOCK  d#2       ;set up a two byte block for num1
0007 0000    num2:    .BLOCK  d#2       ;set up a two byte block for num2
0009 0000    num3:    .BLOCK  d#2       ;set up a two byte block for num3
000B 090003  Main:    LOADA   sum,d     ;load a copy of sum into A
000E E90005           DECI    num1,d    ;read and store a number in num1
0011 190005           ADDA    num1,d    ;add the contents of num1 to A
0014 E90007           DECI    num2,d    ;read and store a  number in num2
0017 190007           ADDA    num2,d    ;add the contents of num2 to A
001A E90009           DECI    num3,d    ;read and store a I number in num3
001D 190009           ADDA    num3,d    ;add the contents of num2 to Ar
0020 80002A           BRLT    Negmsg    ;branch to location NegMsg if negative
0023 110003           STOREA  sum,d     ;store contents of A into sum
0026 F10003           DECO    sum,d     ;output the contents of sum
0029 00      Quit:    STOP              ;stop the processing
002A E00045  Negmsg:  CHARO   c#/E/,i   ;output 'E'
002D E00072           CHARO   c#/r/,i   ;output 'r'
0030 E00072           CHARO   c#/r/,i   ;output 'r'
0033 E0006F           CHARO   c#/o/,i   ;output 'o'
0036 E00072           CHARO   c#/r/,i   ;output 'r'
0039 700029           BR      Quit      ;
003C                  .END              ;end of the program


Symbol        Value           Symbol        Value
Main          000B            Negmsg        002A
Quit          0029            num1          0005
num2          0007            num3          0009
sum           0003
```

7.5.6 具有循环的程序

之前我们编写了一个读入三个数并求和的程序。下面来改变一下这个程序，让它能够读入limit个数并对它们求和。首先读入并存储limit的值。每当读入一个新值后，就从limit减1。当limit为0时，就终止。下面是该算法的伪代码。

```
Read limit
Set sum to 0
While (limit is not zero)
    Read number
    Set sum to sum + number
    Set limit to limit - 1
```

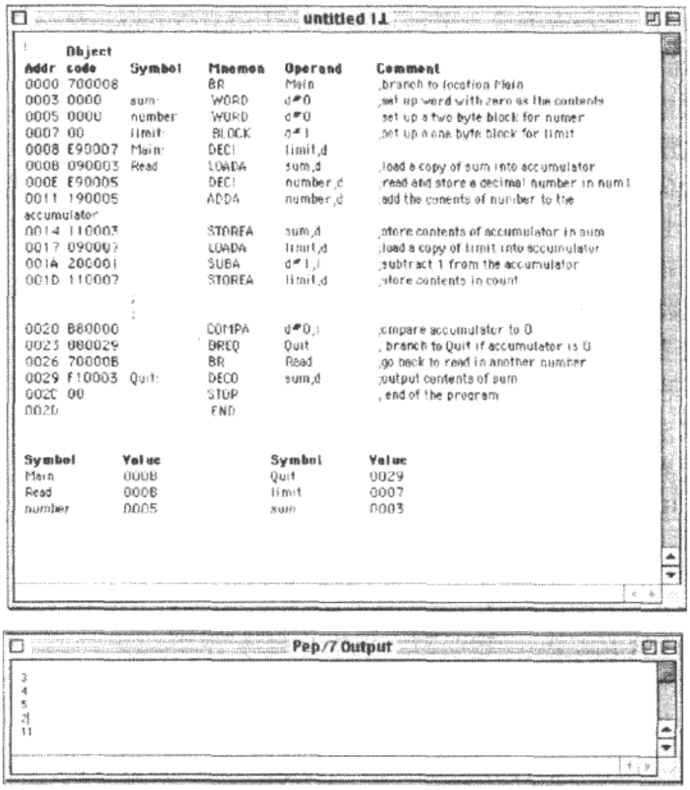
第一步是创建所需的变量，即limit、number和sum。

```
         BR Main
sum:     .WORD d#0     ;set up a word with zero for sum
number:  .BLOCK d#2    ;set up a two byte block for number
limit:   .BLOCK d#1    ;set up a two byte block for limit
```

如何用汇编语言实现limit is not zero这一步呢？使用BREQ指令，如果累加器为0，就把PC设置为操作数。把limit放入累加器，从中减1。然后比较累加器和0，用COMPA指令，设置Z状态位。用BREQ可以测试Z位。如果累加器不为0，则执行读入新值的指令。如果累加器为0，则输出和并退出程序。

```
Set the accumulator to limit
Subtract one from the accumulator
Compare accumulator to zero
if status bit Z is 1
    go to Quit
Else
    go to Read
```

```
Main:    DECI limit, d      ;read and store a decimal number into limit
Read:    LOADA sum,d        ;load a copy of sum into accumulator
         DECI number,d      ;read and store a decimal number in number
         ADDA number ,d     ;add the contents of number to accumulator
         STOREA sum,d       ;store contents of the accumulator into sum
         LOADA limit,d      ;load a copy of limit into accumulator
         SUBA d#1,i         ;subtract 1 from accumulator
         STOREA limit,d     ;store contents in count
         COMPA d#0,i        ;compare accumulator to 0
         BREQ Stop          ;branch to Quit if accumulator is 0
         BR Read            ;go back to read in another number
Quit     DECO sum,d         ;output the contents of sum
         STOP               ;stop the processing
         .END               ;end of the program
```



与具有分支的程序一样，每行代码旁边的注释明确解释了要发生什么。后面的练习将要求你用真实数据手动模拟这些代码。

7.6 其他重要思想

第6章讨论过计算领域中的几个重要思想，即信息隐蔽、抽象、事物命名、程序设计语言和测试。这些思想都出现在本章中，突出了它们之间的纠葛。

7.6.1 抽象

在机器语言层中，几乎没有信息隐蔽，每个细节都必须明确。只有一位信息被隐蔽了，因为Pep/7使用二进制补码表示负数，但在使用机器语言时不必知道这一点。（关于二进制补码，请参阅第2章。）我们不仅可以输入负数并且看到常用的带符号的负数（如−25）结果，还可以在程序中使用负的操作数（如d#−25）。

转移到汇编语言层后，可以用语言自身提供的抽象隐蔽一些细节。例如，可以创建一个存储块，给这个块中的第一个字节一个名字，然后用这个名字来引用这个块。可以给一个字的存储空间分配一个名字，用这个名字把值存入这个字中。可以赋予一条指令一个名字，用这个名字跳转到这条指令。这些关于抽象的例子都涉及给数据或动作命名。

Bug是昂贵的

软件Bug（错误）数不胜数，而且危害非常大，美国商务部的National Institute of Standards and Technology（NIST）的最新研究表明，每年为软件Bug支付的费用约为595亿美元，约占GDP的0.6%。在整个国家范围内，这个花费的一大半由软件用户承担，余下的由软件开发商或销售商承担。[3]

7.6.2　测试

我们测试程序的方法很简单，即执行程序，看它们是否生成了预期的结果。但是，测试远不止运行一次程序这么简单。让我们利用上一个程序，详细地分析一下测试。这个程序将读入三个数字，输出它们的和。设计阶段的测试比较简单，因为设计顶层的每个步骤都是具体步骤。那么，如何测试一个特定的程序以确定它的正确性呢？我们将设计和实现一个**测试计划**。所谓测试计划，就是一个文档，说明了要全面测试程序需要运行的次数以及运行程序使用的数据。每套输入的数据值称为测试用例。测试计划要列出选择这套数据和数据值的原因，还要列出每套数据预期的输出是什么。

测试用例一定要慎重选择。有几种测试方法可以作为测试过程的指导。**代码覆盖法**设计的测试用例会确保程序中的每条语句都能被执行到。因为测试者能够看到代码，所以这种方法又叫做**明箱测试法**。**数据覆盖测试法**是另一种测试方法，它设计的测试用例会确保包括允许使用的数据的边界值。由于这种方法是基于输入的数据，而不是基于代码的，所以它又叫做**暗箱测试法**。常用的测试法是结合这两种方法。

测试计划（test plan）：说明如何测试程序的文档。

代码覆盖（明箱）测试法（code-coverage (clear-box) testing）：通过执行代码中的所有语句测试程序或子程序的测试方法。

数据覆盖（暗箱）测试法（data-coverage (black-box) testing）：把代码作为一个暗箱，基于所有可能的输入数据测试程序或子程序的测试方法。

测试计划实现（test-plan implementation）：用测试计划中规定的测试用例验证程序是否输出了预期的结果。

7.6.3　测试计划实现

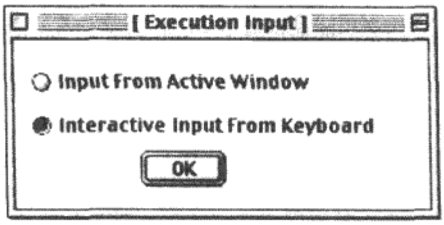
测试计划的实现要运行测试计划中列出的所有测试用例，并记录运行结果。如果结果与预期不符，则必须重新审查设计，找出并纠正其中的错误。当每种测试用例都给出了预期的结果时，这个过程将结束。注意，实现测试计划让我们对程序的正确性有了信心，但可以确定的只是程序对测试用例能够正确地运行。因此，测试用例的质量极其重要。

在读入三个数字并对它们求和这个程序中，明箱测试法只包括三个数据值。这个程序没有测试候选数据的条件语句。但是，仅仅使用明箱测试法是不够的。我们需要尝试正数和负数。读入的数字将存储在一个字中。虽然在问题中没有把数值的范围限制在 $-2^{15}-1$ 和 $+2^{15}-1$ 之间，但是实现却具有这样的限制。在测试计划中，我们应该尝试边界值，不过由于要对它们求和，所以还需要确保它们的和不超出 $-2^{15}-1$ 和 $+2^{15}-1$ 这个范围。

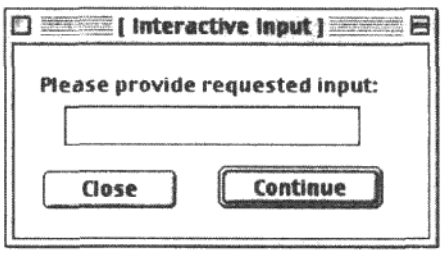
采用该测试用例的原因	输入值	预期的输出	观察到的输出
假设：输入值小于等于 $2^{15}-1$ 并且大于等于 -2^{15}			
输入三个正数	4、6、1	11	
输入三个负数	−4、−6、−1	−11	
输入正负混合的数	4、6、−1	9	
	4、−6、1	−1	
	−4、6、1	3	
大数	32 767、−1、+1	32 767	

要实现这个测试计划，需要运行程序6次，每次采用一套测试用例。结果要记录在“观察到的输出”列中。

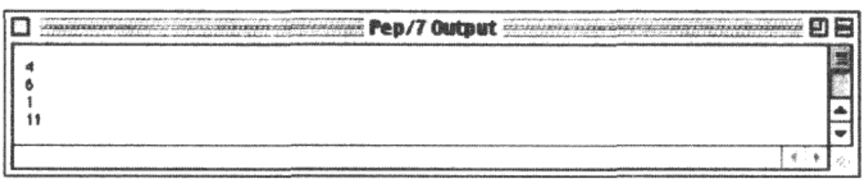
程序既可以从键盘获取输入数据，也可以从文件获取输入数据。键盘输入法叫做交互式输入法。用这种输入法，程序可以在运行时通知用户键入数据。如果选择Pep/7菜单中的Execution Input选项，屏幕上显示的信息如下：



选择Interactive Input From Keyboard单选钮，然后点击OK按钮。在运行程序时，我们将看到下图所示的窗口，要求输入数据。



键入数据后，请点击Continue按钮。这个窗口还会出现两次。如果键入的第一个数是4，第二个数是6，第三个数是1，那么可以得到下图所示的输出。



该窗口显示了程序的输入和输出。第一个测试用例运行正确。本章末的练习将要求你完成这个测试计划的实现。

小结

计算机能够存储、检索和处理数据。用户可以把数据输入计算机，计算机能够显示数据，使用户看到它们。在最底层抽象中，给机器的指令直接反映了这5种操作。

计算机的机器语言是一套机器的硬件能够识别并执行的指令。机器语言程序是一系列用二进制编写的指令。Pep/7是一台具有寄存器A和两部分指令的虚拟计算机。一部分指令说明了要执行的动作，另一部分指令说明了要使用的数据（如果有的话）的位置。Pep/7的模拟程序是行为与Pep/7虚拟机相同的程序，用模拟程序可以运行用Pep/7指令集编写的程序。

Pep/7汇编语言不是使用二进制数表示指令，而是使用助记忆码。用汇编语言编写的程序将被翻译成等价的机器语言程序，然后用Pep/7模拟程序执行。

与算法一样，程序也需要测试。代码覆盖测试法通过仔细检查程序的代码来决定程序的输入。数据覆盖测试法则通过考虑所有可能的输入值来决定程序的输入。

道德问题：软件盗版和版权

你有没有借过朋友的最新软件来更新自己的操作系统？或者只花50美元就买了非常复杂的软件，你会不会忽略自己的怀疑“有这么好的事吗”？对于复制、下载和转卖软件这种行为毫不在乎的态度使软件盗版成了计算机业的一个严重问题。Business Software Alliance所做的一项研究表明，2000年的盗版软件使全球损失了115亿美元。2003年，这个数字达到了290亿美元。美国是全球盗版率最低的国家，但软件公司的收入损失仍然很可观。

所谓软件盗版，就是非法复制取得了版权的软件，或者违反了软件许可中的协议条款。软件许可是列出了用户使用购买的软件需要遵守的条款的文档。如果软件是从朋友那里借的，或者在多台机器上下载了同一个软件，那么你就没有遵守协议，事实上，是违反了法律。

为什么软件要取得版权呢？与一个想法或一个书面作品不同，软件具有功能性。软件这种独特的属性使它有别于其他形式的知识产权，复杂化了它对版权的需求。免费软件基金会的主席Richard Stallman认为，赋予软件版权只会阻碍它的发展，软件的许可费会让许多人对之望而却步。这些负面效应说明，申请普通的版权对软件来说不是最好的方法。开放源代码的拥护者相信，程序的源代码是应该公开的。所谓开放源代码，就是任何人都可以下载的代码，因此，任何人都可以重写程序的某个部分，从而参与到软件的演化进程中。虽然已经有许多程序都是开放源代码的，如LINUX操作系统，但是像Microsoft这样的公司仍然选择保护自己的代码。

如果软件的代码不是开放的，那么从许多方面来看，尊重软件的版权还是很重要的。研究表明，美国每年由于盗版软件问题会丢失107 000份工作。复制朋友的软件带来的危害与下载软件（即在软件没有售出之前就在计算机硬盘上安装未经授权的副本）带来的危害是一样的。使用盗版软件的另一个危害是可能使用户遭受病毒的侵害。从朋友那里“借”软件的用户其实是在剽窃，这种行为会造成严重的后果。

练习

判断练习1～15中的说法的对错：

A. 对　　B. 错

1. 可以在指令寄存器中执行算术运算。
2. 可以在寄存器A中执行算术运算。
3. 可以在累加器中执行算术运算。
4. 如果累加器是0，Z位是1。
5. 如果累加器是负数，N位是0。
6. 程序计数器和指令寄存器是同一个内存单元的两个名字。
7. 寄存器A和累加器是同一个内存单元的两个名字。
8. 指令寄存器的长度是三个字节。
9. 程序计数器的长度是三个字节。
10. 每个状态位的长度是一个字节。
11. 指令说明符的长度是两个字节。
12. 如果要载入累加器的数据存储在操作数中，指令说明符是00。
13. 如果累加器中的数据要存入操作数指定的内存单元，指令说明符是00。
14. 所有的Pep/7指令都占用三个字节。
15. 分支指令将测试状态位。

根据下列（十六进制的）内存状态，找出与练习16～20中的问题匹配的答案。

0001 A2　　0002 11

0003 FF　　0004 00

A. 10100010 00010010　　B. 11111111 00000000

C. 00000000 00000011　　D. 11101101 00000001

E. 00010010 00000000

16. 执行下列指令后，寄存器A中的内容是什么？

00001000 00000000 00000011

17. 执行下列指令后，寄存器A中的内容是什么？

    00001001 00000000 00000011

18. 执行下列两条指令后，寄存器A中的内容是什么？

    00001001 00000000 00000001

    00011000 00000000 00000001

19. 执行下列两条指令后，寄存器A中的内容是什么？

    00001000 00000000 00000001

    00011001 00000000 00000010

20. 执行下列两条指令后，寄存器A中的内容是什么？

    00001001 00000000 00000011

    00100001 00000000 00000010

练习21～53是程序或简答题。

21. 我们说一台计算机是可编程的设备，这句话是什么意思？
22. 列出任何机器语言都必须具备的5种操作。
23. 在算法中，具体步骤和抽象步骤的区别总是不清晰。讨论这个难题，用实例支持你的论点。
24. 什么是虚拟计算机？根据Pep/7计算机讨论这个定义。
25. 我们说过，当指出Pep/7指令一共有32条时，你应该猜出Pep/7的指令是5位的，请解释为什么。
26. 描述这一章中介绍的Pep/7 CPU的特性。
27. 我们只介绍了四种寻址模式中的两种。如果我们没有明说，你会推断出这一点吗？请解释为什么。
28. 如果寻址模式说明符如下，数据（操作数）存放在什么地方？

    a) 00　　　　b) 01

29. 请区分IR（指令寄存器）和PC（程序计数器）。
30. 对Pep/7的内存编址需要多少位？
31. 在不改变指令格式的情况下，需要添加多少内存单元？证明你的答案。
32. 有些Pep/7指令是一元的，只有一个字节。其他指令需要三个字节。根据这一章介绍的指令，定义只需要两个字节的指令是否有用？
33. 如果输入的字符是A，执行下列两条指令的结果是什么？

    0001 11011001 00000000 00000110

    0004 11100000 00000000 00001010

34. 如果输入的字符是A，执行下列两条指令的结果是什么？

    0001 11011001 00000000 00000110

    0004 11100001 00000000 00001010

35. 假设实现语言是Pep/7的机器代码，编写一个算法，输出你的名字。
36. 编写机器语言程序实现练习35中的算法。
37. 假设实现语言是Pep/7的汇编语言，编写一个算法，输出你的名字。
38. 编写汇编语言程序实现练习37中的算法。
39. 用直接寻址模式改写7.4节中的示例程序。
40. 请区别Pep/7菜单中的选项Load、Load/Execute和Execute。
41. 虽然下面的程序可以运行，但是对于某些输入值，会产生奇怪的情况。你能找出其中的bug吗？

```
BR Main
sum:   .WORD d#0
num1:  .BLOCK d#1
num2:  .BLOCK d#1
num3:  .BLOCK d#1
Main: LOADA sum, d
      DECT num1, d
      DECT num2, d
      DECT num3, d
      ADDA num3, d
      ADDA num2, d
      ADDA num1, d
      STOREA sum, d
      DECO sum, d
      STOP
      .END
```

42. 纠正练习41中的错误，运行本章列出的测试计划。
43. 完成正文中执行的读入三个数字并求和这个算法的测试计划。
44. 编写一个算法，读入三个值，输出用第一个值与第三个值的和减去第二个值的结果。
45. 用汇编语言实现练习44中的算法。
46. 为练习45中的程序编写并实现测试计划。

47. 用汇编语言设计并实现一个算法，读入四个值，输出它们的和。
48. 为机器语言程序编写的测试计划适用于同一个解决方案的汇编语言版本吗？请解释你的答案。
49. 请区别伪代码指令.BLOCK和.WORD。
50. 请区别汇编语言的伪代码指令和助记忆码指令。
51. 请区别基于代码覆盖的测试计划和基于数据覆盖的测试计划。
52. 请解释Pep/7菜单中的选项Execution Input的含义。
53. 请为下列指令编写Pep/7汇编语言语句。
    a) 如果累加器为0，跳转到Branch1。
    b) 如果累加器是负数，跳转到Branch1。
    c) 如果累加器是负数，跳转到Branch1；如果累加器不是负数，则跳转到Branch2。

思考题

1. 你喜欢进行汇编语言程序设计吗？认为什么个性的人适合这种繁琐的工作？
2. 我们通过把汇编语言程序翻译成机器语言程序演示了翻译过程。仔细研究练习45的答案。回想汇编器必须执行的步骤。你认为需要查看每条汇编语言指令一次还是两次才能完成翻译操作？试着说服你的朋友自己是正确的。
3. 如果一个人有两台同类型的计算机，那么购买一个软件副本，把它安装在两台机器上是道德的吗？如果回答“是”，论据是什么？如果回答“不是”呢？
4. 将在第12章的传记中出现的Daniel Bricklin没有给他的软件申请专利，因为他相信软件不应该是私有的。结果是他失去了大量版税。你认为他的做法是有远见呢，还是天真？
5. 免费软件基金会是一个免税慈善机构，致力于为GNU工程筹集资金。GNU是一个免费软件。在网络上可以找到他们的观点。比较GNU产品和那些制造商（如Microsoft和Sun）的产品。
6. 如果你打算继续从事与计算相关的工作，成为一名程序员，那么你认为软件应该获得版权还是应该免费？

第8章 高级程序设计语言

第1章介绍过，随着时间的推移，如何围绕硬件建立了不同层次的程序设计语言，简化了应用程序员的工作。第7章先介绍了机器码，然后介绍了用助记忆码（而不是二进制数）表示指令的汇编语言。

虽然汇编语言是前进了一步，但对于不同的机器，程序员仍然需要记住不同的指令。高级程序设计语言与人类的思维和交流方式更接近。由于计算机只能执行机器码，所以需要翻译程序把用高级语言编写的程序翻译成机器码。

这一章将介绍这个翻译过程，然后介绍高级程序设计语言的四种范型，接下来将分析高级语言提供的各种功能。就像“欢迎”这个概念可以用多种语言表达一样，我们描述的功能也可以用不同的语言表达。这里列出了四个具体的高级程序设计语言的例子，即Ada、C++、Java和Visual Basic .NET。

网站上为每种程序设计语言提供了一章，它们是Pascal、Python、C++、Java和VB.NET。读者可能只关注一门语言，而不是这里介绍的所有语言。

目标

学完本章之后，你应该能够：

- 描述翻译过程，区别汇编、编译、解释和执行。
- 列出四种不同的程序设计范型，并说明每种范型的语言特征。
- 描述下列结构：流输入和输出、选择、循环和子程序。
- 构造布尔表达式，并说明如何用它们改变算法的控制流。
- 定义数据类型和强类型化。
- 解释参数的概念，区别值参和引用参数。
- 描述两种复合数据结构机制。
- 列出并描述面向对象语言的三要素，给出每种要素的实例。

8.1 翻译过程

第7章介绍过，用汇编语言编写的程序要输入汇编器，由它把汇编语言指令翻译成机器码，最终执行的是汇编器输出的机器码。使用高级语言，我们要采用其他软件工具协助翻译过程。在研究高级语言之前，先来看看这些工具的基本功能。

8.1.1 编译器

把汇编语言指令翻译成机器码的算法非常简单，因为汇编语言本身就非常简单。所谓简单，指的是每条指令只执行一项基本操作。高级程序设计语言提供的指令集要丰富得多，大大简化了程序员的工作，但由于其中的结构更加抽象，所以翻译过程也难得多。翻译用高级程序设计语言编写的程序的程序叫做**编译器**。早期编译器输出的是程序的汇编语言版本，这

个版本还要经过汇编器处理才能得到可执行的机器语言程序。随着计算机科学家更加深入地了解翻译过程，编译器变得更加复杂，汇编语言的阶段通常被省略了。如图8-1所示。



图8-1　编译过程

编译器（compiler）：把用高级语言编写的程序翻译成机器码的程序。

任何计算机只要具有一种高级语言的编译器，就能运行用这种语言编写的程序。注意，编译器是一种程序，因此，要编译一个程序，就必须具有这个编译器在特定机器上的机器码版本。想要在多种类型的机器上使用一种高级语言，就要具备这种语言的多个编译器。

8.1.2　解释器

解释器是一种翻译程序，用于解释和执行语句序列。与汇编器和编译器只是输出机器码不同的是，解释器在翻译过语句之后会立即执行这个语句。可以把解释器看作编写程序所使用的语言的模拟器或虚拟机。Terry Pratt曾在他关于程序设计语言的经典著作中指出，翻译器和模拟器都接受用高级语言编写的程序作为输入。翻译器（汇编器或编译器）只用适合的机器语言生成等价的程序，这个程序再单独运行。而模拟器则直接执行输入的程序。[1]

解释器（interpreter）：输入用高级语言编写的程序，指导计算机执行每个语句指定的动作的程序。

第二代高级语言可以分为两种，一种是要编译的，一种是要解释的。FORTRAN、COBOL和ALGOL是要编译的语言；Lisp、SNOBOL4和APL是要解释的语言。由于软件解释器非常复杂，所以用要解释的语言编写的程序通常比要编译的程序的运行速度慢很多。因此，要编译的语言发展成了主流，以致产生了Java。

Java是1996年面世的，随后以迅雷不及掩耳之势攻占了整个计算领域。在Java的设计中，可移植性是最重要的特性。为了达到最佳可移植性，Java将被编译成一种标准机器语言——**字节码**。怎么会存在标准机器语言呢？一种名为JVM（Java虚拟机）的软件解释器将接收字节码程序，然后执行它。也就是说，字节码不是某个特定硬件处理器的机器语言，任何具有JVM的机器都可以运行编译过的Java程序。

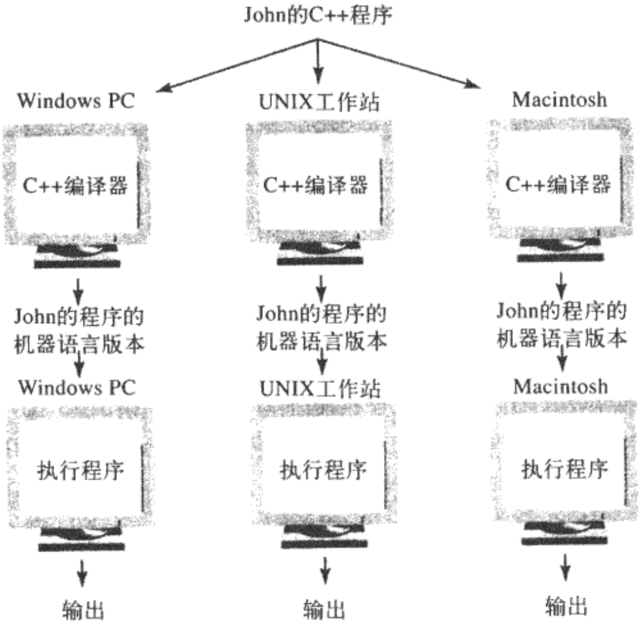
字节码（bytecode）：编译Java源代码使用的标准机器语言。

注意，标准化的高级语言实现的可移植性与把Java程序翻译成字节码然后在JVM上解释它所实现的可移植性是不同的。用高级语言编写的程序能够在任何具有适合的编译器的机器上编译和运行，程序将被翻译成计算机能够直接执行的机器码。而Java程序则是被编译成字节码，编译过的字节码程序可以在任何具有JVM解释器的机器上运行。也就是说，Java编译器输出的程序将被解释，而不是被执行。请参阅图8-2。Java程序总是被翻译成字节码。此外，还有一些语言的编译器是把语言翻译成字节码，而不是翻译成机器码的。例如，Ada编译器就是把Ada语言翻译成字节码。

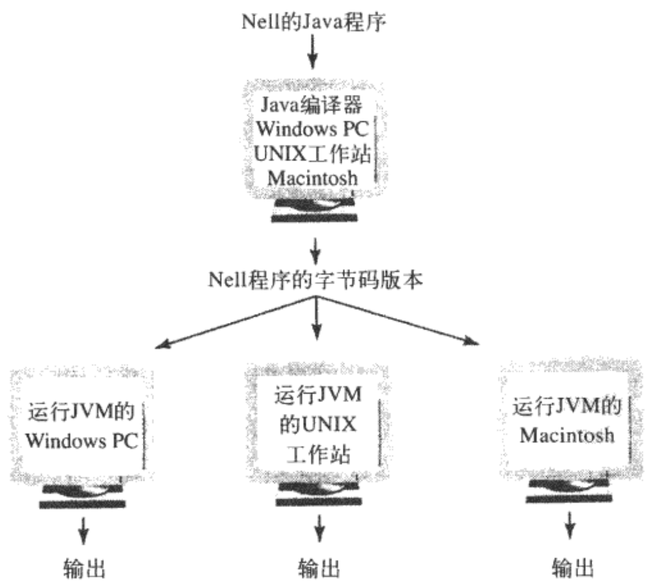
JVM是像第7章讨论过的Pep/7一样的虚拟机。我们将虚拟机定义为用于说明真实计算机的重要特性的假想机。JVM是为执行字节码程序设计的假想机。

UCSD的p-系统早于字节码

p-代码是一种类似于字节码的语言，美国加州大学圣迭戈分校（UCSD）在20世纪70年代开发了一个执行p-代码的系统。用Pascal和FORTRAN编写的程序将被翻译为p-代码，任何具有p-代码解释器的机器都可以执行这些翻译过的程序。



a) 在不同系统上编译和运行的C++程序



b) 编译成字节码的Java程序在不同的系统上运行

图8-2 标准化的语言提供的可移植性和解释字节码提供的可移植性

8.2 程序设计语言的范型

什么是范型？《美国传统词典》对范型的定义有两条与计算相关，即“用作模式或模型的实体”和“一组假设、概念、数值和规则，构成了共享它们的聚合体观察现实的方式，尤其适用于精神学科。”[2]第1章概述了软件的发展史，其中列出了每个时代开发的程序设计语言。另一种观察程序设计语言的方法，是看不同语言反映现实的不同方面的方式，也就是说，看表示它们的范型。

范型一词的变迁

在1977年的《韦氏新大学词典》（Webster's New Collegiate Dictionary）中，范型被定义为“一个实例或模式、一个原型的显著实例或者一个词的变形或格变化的实例。”这里没有提及聚合体。2006年在Internet上检索时，发现了许多相关的定义，包括“一种模式或实体的实例。”这个词还暗含意象和思维模式的含义。Thomas Kuhn用这个词表示科学家掌握特定领域的知识的模型。Kuhn的著作《The Structure of Scientific Revolutions》描述了他划分的科学从一个范型发展到另一个范型的各个阶段。[3]“他提出了心理分析模型中的问题。”[4]“一种模型或引用框架。看待一个观点或问题的态度的激烈转变。”[5]

对存储在内存中的数值进行操作的顺序指令的模型是冯·诺伊曼模型，这种模型极大地影响了程序设计语言最常用的模型——*命令模型*（或称为*流程模型*）。整个计算软件史中的主要语言采用的都是这种范型。这些语言包括FORTRAN、COBOL、BASIC、C、Pascal、Ada和C++。使用这种范型的语言允许程序员把算法表示为第6章介绍的任务分层体系。也就是说，程序描述了解决问题的必要处理。命令范型的特征是顺序执行指令，使用表示内存地址的变量，以及使用赋值语句改变这些变量的值。[6]

另一种计算模型是*函数模型*，它以数学概念函数为基础。计算被表示为函数求值。问题求解被表示为函数调用。基本结构是函数求值，不存在变量和赋值语句。例如，两个值的加法可以表示为：

(+30 40)

其中括号表示一个表达式，把第一项（必须是函数）应用到列表中的其余值是对表达式求值。因此，这个表达式是把加法函数应用到接下来的两个数字来求值的，返回的是数值70。没有循环结构，重复操作被表示为递归的函数调用（本章后面的小节会讨论递归）。最著名的使用函数范型的语言是LISP、Scheme（由LISP派生的语言）和ML。

*逻辑程序设计*是第三种程序设计范型。逻辑程序设计的基础是数理逻辑的原理。该模型由一组关于对象的事实和一组关于对象之间的关系的规则构成。采用这种模型的程序，由关于对象和它们之间的关系（可以由事实和规则推出）的问题构成。采用这种模型，底层的问题求解算法使用逻辑规则由事实和规则推导出答案。

PROLOG是第三代逻辑程序设计语言，于1970年在法国开发出来。直到1981年它才凸显出来，因为当时日本宣称逻辑程序设计将在他们的第五代计算机中扮演重要角色。PROLOG程序由三种类型的语句构成。一种语句用于声明有关对象的事实和对象之间的关系。一种语句用于定义有关对象和它们之间的关系的规则。第三种语句用于对对象和它们的关系发问。[7]

第13章要介绍的人工智能应用程序既使用了LISP，又使用了PROLOG。你会发现，用这

些语言编写的程序，与采用命令范型的语言所反映的冯·诺伊曼体系结构毫无相似之处。

第四种范型是面向对象范型。面向对象的观点将世界看作由交互的对象构成。每个对象负责自己的动作。在命令范型中，数据对象是被动的，由程序进行操作。在面向对象范型中，对象则是主动的。对象和操作对象的代码绑定在一起，这使得每个对象都负责它自己的操作。采用面向对象范型的语言允许程序员用（第6章所述的）对象分层体系表示算法。

SIMULA和Smalltalk是最早的两种面向对象程序设计语言。虽然有些人认为C++是面向对象的语言，但是我们认为它是具有某些面向对象特性的命令式语言。而Java是具有某些命令式特性的面向对象语言。

命令式和面向对象是最常用的软件开发范型，也是本章的重点。

8.3 命令式语言的功能性

第6章介绍了通用的问题求解方法以及在计算机上实现的问题求解方案，其中提到过两种算法设计使用的过程——选择和迭代（循环）。但在汇编语言中却没有实现这两种结构。虽然它们是可以实现的，但是实现起来非常繁琐，超出了本书要介绍的有关汇编语言的内容范围。在高级语言中，选择和迭代操作则非常简单。首先，我们将介绍布尔表达式的概念，它是高级语言用于进行选择的结构。然后，我们将分析高级语言提供的使程序设计更容易、更安全的结构。

8.3.1 布尔表达式

第6章编写了一个读取数对并按序输出它们的算法。让我们看看这个算法。

```
Write "How many pairs of values are to be entered?"
Read numberOfPairs
Set numberRead to 0
while (numberRead < numberOfPairs)
    Write "Enter two values separated by a blank; press return"
    Read number1
    Read number2
    If (number1 < number2)
        Print number1 + " " + number2
    Else
        Print number2 + " " + number1
    Increment numberRead
```

该算法包括一个循环和一个选择语句。在输入的数对没有到达要求的数量时，循环将继续执行。如果第一个数小于第二个数，就按照这个顺序输出它们；否则，按照相反的顺序输出它们。注意这些问题是用短语表达的：

```
(numberRead < numberOfPairs)
(number1 < number2)
```

每个短语实际上是一个语句。如果这个语句是true，那么这个问题的答案就是true。如果语句不是true，那么这个问题的答案就是false。写出语句，然后测试它们是true还是false，这

是程序设计语言提问的方式。这些语句称为断言或条件。在编写算法时，我们采用自然语言表示断言。在把算法翻译为高级程序设计语言时，这种用自然语言编写的语句将被重写为布尔表达式。

什么是布尔表达式？第4章在讨论门和电路时介绍过布尔运算。这里将它们应用在逻辑层，而不是硬件层。**布尔表达式**是一个标识符序列，标识符之间由相容的运算符分隔，求得的值是true或false。一个布尔表达式可以是：

- 一个布尔变量。
- 一个算术表达式加一个关系运算符，再加一个算术表达式。
- 一个布尔表达式加一个布尔运算符，再加一个布尔表达式。

迄今为止，在示例中，变量存放的都是数值。布尔变量是内存中的一个地址，由存放true或false的标识符引用。

> **布尔表达式**（Boolean expression）：一个标识符序列，标识符之间由相容的运算符分隔，求得的值是true或false。

关系运算符是比较两个值的运算符。下表总结了六种关系运算符以及各种高级语言用于表示它们的符号。

符　号	含　义	示　例	计算法则
<	小于	Number1<Number2	如果Number1小于Number2，为true，否则为false
<=	小于等于	Number1<=Number2	如果Number1小于等于Number2，为true，否则为false
>	大于	Number1>Number2	如果Number1大于Number2，为true，否则为false
>=	大于等于	Number1>=Number2	如果Number1大于等于Number2，为true，否则为false
!=或<>或/=	不等于	Number1!=Number2	如果Number1不等于Number2，为true，否则为false
=或==	等于	Number1==Number2	如果Number1等于Number2，为true，否则为false

两个算术表达式之间的关系运算符是询问两个表达式之间是否存在这种关系。例如：

```
xValue < yValue
```

是一个断言，即xValue小于yValue。如果xValue确实小于yValue，那么这个表达式的结果是`true`；如果xValue大于或等于yValue，那么结果为`false`。

布尔运算符是特殊的运算符AND、OR和NOT。如果两个表达式都是`true`，AND运算符就返回`true`，否则返回`false`。如果两个表达式都是`false`，OR运算符就返回`false`，否则返回`true`。NOT运算符将改变表达式的值。这些运算与第4章中介绍的功能性一致。在硬件层中，我们指的是电流状态和单独位的表示法。在这一层中采用的还是相同的逻辑，只是我们指的是语句的对或错。

8.3.2　强类型化

在使用汇编语言时，标识符被赋予了内存单元，标识符与这些内存单元存储的内容无关。然而，大多数高级语言在关联内存单元和标识符时则要求说明这些内存单元存储的数据类型。如果程序中的某个语句要把类型不符的值存入一个变量，就会出现错误消息。只有类型相符的值才能存入变量的要求被称为**强类型化**。

接下来的几节将介绍几种常用的数据类型以及高级语言如何把内存单元与标识符关联起

来。每种数据类型都有某些可以应用的合法操作。所谓**数据类型**，是一组值以及能够应用于这种类型的值的基本操作集合的说明。

> **强类型化**（strong typing）：每个变量都有一个类型，只有这种类型的值才能存储到该变量中。
> **数据类型**（data type）：一组值以及能够应用于这种类型的值的基本操作集合的说明。

数据类型

数据是表示信息的物理符号。在计算机内部，数据和指令都是二进制位的组合。计算机能够执行一条指令，是因为这条指令的地址被载入了程序计数器，指令被载入了指令寄存器。被执行的位组合同样可以表示整数、实数、字符或布尔值，关键看计算机如何解释位组合。

例如，Pep/7的Stop指令是一个所有位为0的字节。当这条指令被载入指令寄存器后，程序将停止。一个所有位为0的字节也可以解释为一个值为0的8位二进制数。如果一个所有位都是0的内存单元被加到了一个寄存器的内容上，那么这个内存单元中的值将被解释为一个数字。

> **由单词bow想到的**
>
> 单词是字符表中的符号序列。有些符号序列或组合被赋予了含义，但有些则没有（例如，在英语中，符号序列ceba并没有意义）。bow是一个英语单词，但它有多种含义，如船首、小姑娘佩戴的蝴蝶结、演奏小提琴用的琴弓或者鞠躬的动作。根据这个单词的上下文，可以分辨出它的含义，同样地，编译器也能够根据一个单词周围的语法分辨它的含义。

大多数高级语言都固有四种数据类型，即整数、实数、字符和布尔值。

整数

整数数据类型表示的是一个整数范围，这个范围由表示整数值的字节数决定。有些高级语言提供几种范围不同的整数类型，允许用户根据特定问题选择适合的类型。

应用于整数的操作是标准的数学运算和关系运算。加法和减法由标准符号+和－表示。乘法和除法通常表示为*和/。不同语言的整数除法返回的结果不同，有的返回一个实数，有的则返回商。有的语言有两个符号用于除法，一个返回实数，一个返回商。大多数语言还有一个返回整数除法的余数的运算，称为*模运算*，不过这个运算可能是数学中的模运算，也可能不是。关系运算符由上一节列出的关系运算符表中的符号表示。

实数

实数数据类型表示的是特定精度的数的范围，与整数数据类型一样，这个范围由表示实数值的字节数决定。许多高级语言有两种实数。应用于实数的操作与应用于整数的一样。但在对实数应用关系运算时要小心，因为实数通常不精确。例如，在计算机上计算1/3+1/3+1/3并不等一定于1.0，实际上，1/10*10也不一定等于1.0。

字符

第3章介绍过，表示ASCII字符集中的字符需要一个字节，表示Unicode字符集中的字符则需要两个字节。ASCII字符集包括英语字符，是Unicode字符集的子集。对字符进行数学运算是毫无意义的，许多强类型化的语言都不允许进行这种运算。但比较字符却是有意义的，所

以可以对字符进行关系运算。在字符的关系运算中，“小于”和“大于”的意思是这个字符在字符集中“在……之前”和“在……之后”。例如，字符‘A’小于‘B’，字符‘B’小于‘C’，等等。字符‘1’小于字符‘2’，‘2’小于‘3’，等等。如果要比较‘A’和‘1’，必须在使用的字符集中查找这两个字符的关系。

布尔型

如上一节所述的，布尔数据类型只有两个值——true和false。并非所有的高级语言都支持布尔数据类型。如果一种语言不支持布尔类型，可以模拟它，用1表示true，用0表示false。

整数、实数、字符和布尔称为简单数据类型或原子数据类型，因为每个值都是独立的，不能再分割的。8.3.5节将讨论复合数据类型，即由一组值构成的数据类型。字符串是一种具有复合数据类型的特征的数据类型，但通常被看作简单数据类型。

字符串

字符串是一个字符序列，这个序列通常被看作一个值。例如，

```
"This is a string."
```

是一个字符串，包含17个字符，分别是1个大写字母、12个小写字母、3个空格和1个句号。不同语言定义的字符串的操作不同，包括连接操作和根据词典顺序进行的比较操作。有些语言提供了一组完整的操作，如提取子串或检索子串等。

注意，我们使用单引号圈起字符，用双引号圈起字符串。有些高级语言采用同样的符号圈起字符和字符串，因此一个字符和只包含一个字符的字符串之间没有区别。

声明

声明是把变量、动作或语言中的其他实体与标识符关联起来的语句，使程序员可以通过名字引用这些项目。这一节将讨论如何声明变量，然后介绍如何命名动作。

> **声明**（declaration）：把变量、动作或语言中的其他实体与标识符关联起来的语句，使程序员可以通过名字引用这些项目。

下表展示了在三种不同的语言中如何声明四个相同的变量：

语　言	变量声明
Ada	`sum : Float : = 0; - - set up word with 0 as contents` `num1: Integer;     - - set up a two byte block for num1` `num2: Integer;     - - set up a two byte block for num2` `num3: INTEGER;     - - set up a two byte block for num3` `. . .` `num1:= 1;`
VB.NET	`Dim sum As Single = 0.0F ' set up word with 0 as contents` `Dim num1 As Integer ' set up a two byte block for num1` `Dim num2 As Integer ' set up a two byte block for num2` `Dim num3 As Integer ' set up a two byte block for num3` `. . .` `num1 = 1`

（续）

语 言	变量声明
C++/Java	`float sum = 0.0;  // set up word with 0 as contents` `int num1;         // set up a two byte block for num1` `int num2;         // set up a two byte block for num2` `int num3;         // set up a two byte block for num3` `. . .` `num1 = 1;`

这些例子说明了高级语言中的一些不同之处。VB.NET采用了一个保留字来标示声明。**保留字**是一种语言中具有特殊意义的字，不能用它作为标识符。Dim是VB.NET中用于声明变量的保留字。Ada、C++和Java声明变量时不采用保留字。Ada、C++和Java的语句结尾使用的是分号，VB.NET则使用行结束符或注释符号。这些语言采用的注释符号各不相同。Pep/7使用分号标示接下来的是注释。

注意，在Ada中，标示整数类型的保留字是大小写混用的。Ada是不**区分大小写**的语言，也就是说，大写和小写是一样的。C++、Java和VB.NET则区分大小写。同一个标识符，如果大小写形式不同，则表示不同的字。因此，Integer、INTEGER、InTeGeR和INTeger在Ada中将被看作一个标识符，而在C++和VB.NET中则被看作四个不同的标识符。在C++和Java中，保留字int表示整数，float表示单精度实数。Ada用Float表示实数，而VB.NET则分别用Single和Double表示实数的两种版本。

> **保留字**（reserved word）：一种语言中具有特殊意义的字，不能用它作为标识符。
>
> **区分大小写**（case sensitive）：大写字母和小写字母被看作是不同的；两个拼写方法相同，但大小写形式不同的标识符被看作两个不同的标识符。

每种语言都允许在声明之后加上赋值运算符和值，从而在分配给这个标识符的内存单元中存放一个初始值。像伪代码一样，用赋值语句可以把一个值存入变量。Ada用:=作为赋值运算符；VB.NET、C++和Java使用的是=。

这些区别重要吗？如果用其中一种语言编写程序，这些区别的确非常重要。然而，这些只是语法问题，即做同一件事的不同方式。真正重要的概念是标识符与一个内存单元关联在一起，具有特定的数据类型。在练习中，我们将要求你对比这些示例中的语法区别。

> **赋值语句**（assignment statement）：把一个表达式的值存入一个变量的语句。

标识符大小写的用法是语言文化的一部分。在示例中，我们尽量与常见的语言文化保持一致。例如，大多数C++程序员用全大写字母表示有名常数，变量名的开头使用小写字母，而Ada和VB.NET程序员则用大写字母作为变量名的第一个字母。

8.3.3 输入/输出结构

在算法的伪代码中，我们曾用Read和Write或Print表达式说明在与程序以外的环境交互。Read表达式负责从外部环境获取一个值，并将其存入程序内的变量。Write或Print表达式负责向人们显示消息。

高级语言把输入的数据看作一个分为多行的字符流。字符的含义则由存放值的内存单元的数据类型决定。所有输入语句都由三部分组成，即要存放数据的变量的声明、输入语句和要读入的变量名以及数据流自身。例如，下面这个输入三个值的算法：

```
Read name, age, hourlyWage
```

在大多数高级语言中，需要分别声明变量name、age和hourlyWage的数据类型。假设它们的数据类型分别是字符串、整数和实数。输入语句将列出这三个变量。处理过程如下。因为name是字符串型的，所以读入操作将假定输入流中的第一个数据项是字符串。这个字符串将被读入并存储到name中。接下来的变量是一个整数，所以读入操作预计输入流中的下一项是一个整数。这个值将被读入并存储在age中。第三个变量是实数，所以读入操作预计输入流中的下一项是一个实数。

输入流可能来自键盘，也可能来自一个数据文件，不过处理过程是一样的。变量出现在输入语句中的顺序必须与值出现在输入流中的顺序一样。输入的变量的类型决定了如何解释输入流中的字符。也就是说，输入流只是一系列ASCII（或Unicode）字符。下一个值要存入的变量的类型决定了如何解释这个字符序列。为了便于叙述，假设输入语句采用空格分隔每个数值。例如，假设数据流如下：

```
Maggie 10 12.50
```

"Maggie"将被存储到name中；10将被存储到age中；12.50将被存储到hourlyWage中。10和12.50都是作为字符被读入的，然后被分别转化成整数和实数。

输出语句将创建一个字符流。输出语句中列出的项目可以是直接量，也可以是变量名。直接量是输出语句中明确写出的数字或字符串。根据标识符或直接量的类型，输出的值将被依次处理。类型决定了如何解释这种位组合。如果类型是字符串，写入输出流的就是字符。如果位组合是一个数，这个数将被转化成表示其中数字的字符序列，写入输出流的是这些字符。

无论输入/输出语句的语法是什么，也无论输入/输出流在哪里，这个过程的关键是数据类型，它决定了如何把字符转化成位组合（输入）以及如何把位组合转化成字符（输出）。

这里没有给出输入/输出语句的示例，因为它们的语法通常非常复杂，而且在不同高级语言中，它们的语法有很大差别。

8.3.4 控制结构

伪代码有两种方式来改变算法的控制流，即循环和选择。这些结构叫做控制结构，因为它们决定了程序中指令的执行顺序。

> **控制结构**（control structure）：确定程序中的其他指令的执行顺序的指令。

Edsger W. Dijkstra在1972年发表的论文“Notes on Structured Programming”中指出，程序员应该是严格的并遵守规则的，他们只应使用选定的控制结构。这篇论文和其他同期发表的论文开创了*结构化程序设计*的时代。[8]程序中的每个逻辑部件都只能有一个入口和一个出口，程序不应随意地跳入或跳出这些逻辑模块。虽然在汇编程序中可以用分支语句这样跳转，但高级语言引入的控制结构使得这一规则比较容易遵守。这些控制结构是选择语句、循环语句和子程序语句。无限制的分支语句不再是必需的。

在关于伪代码的讨论中，我们考察了选择和循环语句。那时，在设计中，我们没看到过子程序语句，即给出来一个任务名，但却没有展开实现这个任务。使用子程序语句，就可以把子程序名放在代码中，然后在代码的另一个地方放实现任务的代码。

随着视窗和鼠标输入方式的出现，高级语言引入了第五种控制结构——异步处理。下面的几节将从逻辑层出发介绍这些概念。

Edsger Dijkstra[9]



在人们付出努力的每个领域中，都有其杰出贡献者，这些人由于他们的理论洞察力、对基本想法的扩展或对某一学科的革新而为人们所称道。像古典音乐世界中的Beethoven、Schubert、Mozart和Hayden，像摇滚音乐界的Beatles和Rolling Stones，Edsger Dijkstra在计算机语言的殿堂中占有重要位置。

Dijkstra于1930年出生在荷兰鹿特丹的一个药剂师家中，他对于形式主义有种特殊的偏好。在荷兰的莱顿大学学习期间，他参加了英国剑桥大学举办的程序设计课程，开始对程序设计着迷。1952年，他在阿姆斯特丹的Mathematical Centre进行兼职，毕业之后继续在这里工作。20世纪70年代初，他作为Burroughs公司的研究员来到美国。1984年9月，他来到得克萨斯大学奥斯汀分校，开始担任计算机科学系的Schlumberger Centennial Chair，于1999年11月退休。

Dijkstra对程序设计最著名的贡献之一是极力提倡结构化程序设计原则。Dijkstra发现，运行具有goto语句的程序就像进入了老鼠洞，常常会杂乱无章地在程序各个部分之间跳来跳去，即使是作者本人，也难以理解自己的程序，更不要说以后要维护它的同事了。Dijkstra认为，goto语句对控制结构来说可有可无，他强烈推荐使用迭代或循环这样的结构，明确地用括号括起程序分支的作用域，有效地对程序进行自注解。Dijkstra声称，坚持结构化程序设计原则可以使程序更容易理解和维护，减少出错的机会。

撇开他的理论贡献不提，Dijkstra在计算领域可谓是个有趣的角色。他表达自己思想的方式非常狂放且极具煽动性，以至于大多数人都不能侥幸逃脱他的奚落。例如，Dijkstra曾经评论道“使用COBOL语言会使人变傻，所以教授COBOL语言应被看作一种犯罪行为。”不止一种语言受到过他的批评，他还说过“如果一个学生学习过BASIC，那么就不可能让他学好程序设计，这样的程序员智力受到了损毁，没希望复原了。”有些人认为他的说法令人信服，觉得他的态度是说服别人赞成自己的观点所必需的。而另外一些人则十分清楚程序设计语言的发展史以及每种语言的设计背景，因此，尽管欣赏他的言论，但觉得他的态度太过尖刻了。

除了对程序设计语言的贡献，Dijkstra还以证明程序正确性的工作而闻名。程序正确性这个领域是把数学应用于计算机程序设计。研究者们致力于构造一种语言和证明方法，用于证明一个程序能无条件地按照它的规约执行，完全没有bug。毫无疑问，无论应用程序是给客户开账单的系统，还是飞行控制系统，这一主张都极其有用。

1972年，ACM为了表彰Dijkstra对这个领域的杰出贡献，授予了他著名的图灵奖。引用的一段颁奖词如下：

“ALGOL是一种高级程序设计语言，现在已经成了清晰性和数学精确性的模型，在

20世纪50年代后期开发ALGOL的过程中，Edsger Dijkstra是主要的贡献者。总的来说，他是程序设计语言这门科学和艺术的主要解释者之一，为我们理解程序设计语言的结构、表示和实现做出了极大的贡献。他于15年间发表的作品包罗万象，包括关于图论的理论性文章、基本手册、注释性文档和程序设计语言领域的哲学构思。"

1989年，SIGCSE（Special Interest Group for Computer Science Education）授予了他计算机科学教育杰出贡献奖。

当Dijkstra发现自己只能再活几个月时，他和妻子返回了荷兰。虽然他想在奥斯汀退休，但却于2002年8月6日在荷兰逝世了。

2003年3月，计算界各位同仁收到了下面的电子邮件：

在此宣布把"PODC Influential-Paper Award"更名为"Edsger W. Dijkstra Prize in Distributed Computing"，以纪念分布式计算领域的先驱Edsger W. Dijkstra。他对并发元素（如信号量）、并发问题（如互斥和死锁）、并发系统的推理和自稳定性的基础研究是分布式计算领域最强有力的支持。其他人对分布式计算规则的研究的影响都没有这么大。

http://www.podc.org/dijkstra/上有关于Dijkstra获得的奖项的信息。

Dijkstra有很多简洁幽默的名言。其中所有研究计算的人都应该细细思量的一句话是"计算机科学不过是关于计算机的科学，就像天文学不过是关于望远镜的科学一样"。

顺序

一条指令按照物理顺序排在另一条指令之后执行，这仍然是命令式程序的基本结构。如果没有遇到改变指令执行顺序的指令，那么语句将一直按照顺序执行。

选择语句

如第6章所述，if语句有两个版本。一个版本（if-then）是执行一组语句或跳过它们。另一个版本（if-then-else）则是执行两组语句中的一组。

让我们看一个具体的例子。如果程序中的一个变量表示温度（temperature），那么我们来比较温度的值，以确定穿什么样的衣服。首先用伪代码陈述这个算法，然后看看如何把它转化成高级语言编写的程序。

```
If (temperature > 75)
  Write "No jacket is necessary"
Else
  Write "A light jacket is appropriate"
```

下表说明了Ada、VB.NET、C++和Java如何实现这个算法。

语　言	if 语 句
Ada	`if Temperature > 75 then` `  Put(Item => "No jacket is necessary")` `else` `  Put (Item => "A light jacket is appropriate");` `end if;`
VB.NET	`if (Temperature > 75) Then` `  MsgBox("No jacket is necessary")`

（续）

<table>
<tr><th>语　言</th><th>if 语 句</th></tr>
<tr><td>VB.NET</td><td><pre>Else
  MsgBox("A light jacket is appropriate")
End if</pre></td></tr>
<tr><td>C++</td><td><pre>if (temperature > 75)
  cout << "No jacket is necessary";
else
  cout << "A light jacket is appropriate";</pre></td></tr>
<tr><td>Java</td><td><pre>if (temperature > 75)
  System.out.print("No jacket is necessary");
else
  System.out.print("A light jacket is appropriate");</pre></td></tr>
</table>

在伪代码中我们通过缩进对语句进行分组。高级语言都有把指令序列作为一个组处理的方法。下面的例子给else分支添加了第二个输出语句，让我们看看这四种语言如何处理这种情况。

<table>
<tr><th>语　言</th><th>if 语 句</th></tr>
<tr><td>Ada</td><td><pre>if Temperature > 75 then
  Put(Item => "No jacket is necessary");
else
  Put (Item => "A light jacket is appropriate");
  Put (Item => "but not necessary");
end if;</pre></td></tr>
<tr><td>VB.NET</td><td><pre>if (Temperature > 75) Then
  MsgBox("No jacket is necessary")
Else
  MsgBox("A light jacket is appropriate")
  MsgBox("but not necessary")
End if</pre></td></tr>
<tr><td>C++</td><td><pre>if (temperature > 75)
  cout << "No jacket is necessary";
else
{
  cout << "A light jacket is appropriate";
  cout << "but not necessary";
}</pre></td></tr>
<tr><td>Java</td><td><pre>if (temperature > 75)
  System.out.print("No jacket is necessary");
else
{
  System. out. print ("A light jacket is appropriate");
  System.out.print("but not necessary");
}</pre></td></tr>
</table>

被看作一组的指令称为*复合语句*。Ada和VB.NET用保留字结束一个结构，因此不需要额外的分组标记。C++和Java用花括号对语句分组，它们称这些分组为块。

如果布尔表达式的值不是true，只会执行else分支，因此可以利用这条特性提一系列问题。让我们细化一下前面有关温度的算法。

```
If (temperature > 90)
  Write "Texas weather: wear shorts"
else If (temperature > 70)
  Write "Ideal weather: short sleeves are fine"
else if (temperature > 50)
  Write "A little chilly: wear a light jacket"
else If (temperature > 32)
  Write "Philadelphia weather: wear a heavy coat"
else
  Write "Stay inside"
```

到达第二个if语句的唯一方法是第一个表达式的值为false，因此，如果第二个表达式的值是true，就可以知道温度在华氏71～90度之间。如果第一个表达式和第二个表达式都是false，而第三个表达式是true，那么温度将在51～70度之间。采用同样的推理方法，可以得出Philadelphia的温度介于33～50度之间。如果温度低于或等于32度，则显示“Stay inside”（留在户内）的提示。任何一个分支都包括一个语句序列。

循环语句

第6章中介绍过重复执行一个语句序列的概念。在这一章前面重复的子算法中有下列表达式：

```
While (more pairs)
```

其后的处理过程将被反复执行，直到所有的name都处理过为止。也就是说，只要这个表达式的值是true，语句序列就会被重复执行。当表达式的值变为false后，将执行紧接着循环的语句。while语句和if语句一样，改变了程序正常的顺序流。

注意，if语句用于在两组动作中选择其一；while语句则用于重复一组动作。图8-3展示了if语句的行为。图8-4展示了while语句的行为。

在介绍不同的高级语言如何表示while语句前，我们来看看两种不同的循环类型。

计数控制的循环　计数控制的循环将重复执行指定的次数。这种循环机制每当过程重复一次，就累计一次重复次数，然后在再次开始重复操作之前，测试循环次数，看循环是否结束了。这种循环由三部分构成，其中用到了一种特殊变量——*循环控制变量*。第一部分是初始化，循环控制变量将被初始化为某个起始值。第二部分是测试，测试循环控制变量是否达到了预定的值。第三部分是增量，循环控制变量将被加1。下面的算法将重复一个过程limit次。

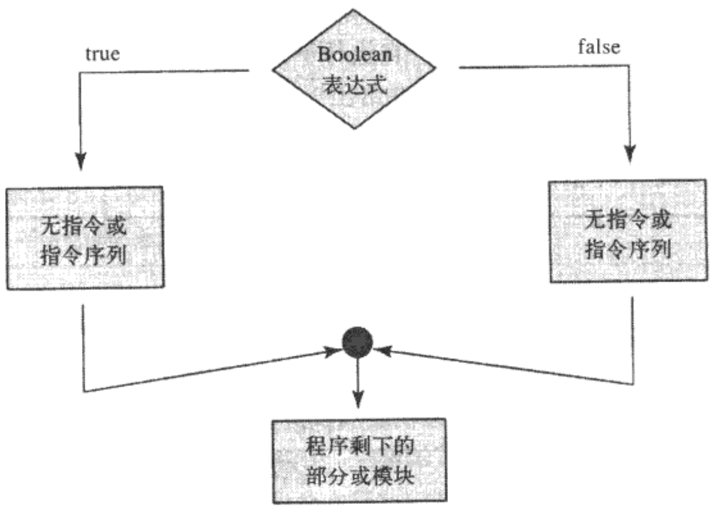


图8-3 if语句的控制流

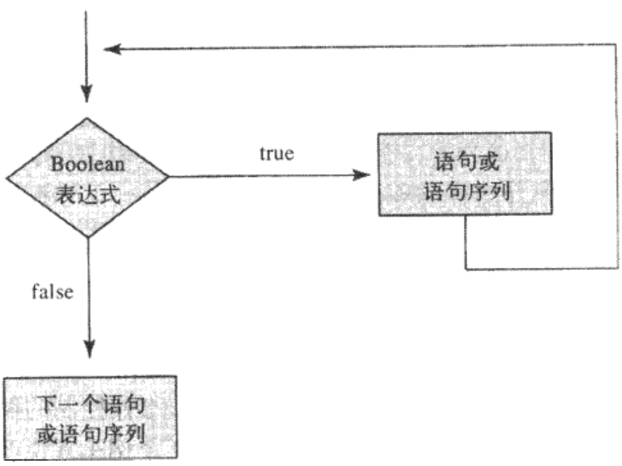


图8-4 while语句的控制流

Set count to 1	把count初始化为1
While (count <= limit)	测试
...	循环主体
Set count to count + 1	增量
...	循环之后的语句

count是循环控制变量，在循环之外被设置为1。While语句将测试表达式count<=limit的值，只要这个表达式的值是true，循环主体就会被执行。循环中的最后一个语句增加了循环控制变量count的值。这个循环将执行多少次呢？当count值为1, 2, 3, …limit时，循环都要执行，所以这个循环要执行limit次。循环控制变量的初始值和布尔表达式中的关系运算决定了循环执行的次数。

while循环被称为预先测试循环。也就是说，测试发生在循环执行之前。如果初始条件是false，那么将不会进入循环。如果漏掉了增量操作会出现什么情况呢？布尔表达式将永远不

会发生变化。如果表达式的值最初就是false，那么什么都不会发生。如果表达式最初是true，而表达式又绝对不会改变，那么循环将一直执行下去。不过大多数计算系统都有定时装置，所以程序是不会永远运行的，它会中断，并给出一个错误消息。永远不会终止的循环叫做*无限循环*。

下表展示了用Ada、VB.NET、C++和Java实现的while语句。

语　言	采用while语句的计数控制循环
Ada	`Count := 1;` `while Count <= Limit loop` `  ...` `  Count := Count + 1;` `end loop;`
VB.NET	`Count = 1` `While (count <= limit)` `  ...` `  count = count + 1` `End While`
C++/Java	`count = 1;` `while (count <= limit)` `{` `  ...` `  count = count + 1;` `}`

事件控制的循环　循环次数由循环主体中发生的事件控制的循环叫做事件控制的循环。用while语句实现的事件控制的循环也具有三个部分。必须初始化事件、测试事件和更新事件。

计数控制循环非常直观，过程将被执行指定的次数。事件控制的循环就不那么明确。究竟事件应该是怎样的，你可能不会立刻明白，让我们用两个例子来说明。首先，我们来读入一组数据，并计算它们的和，直到读入了一个负数为止。这个例子中的事件是什么？是输入值为正数。如何初始化这个事件呢？读入第一个数值即可。测试这个值，可以确定它是否为正数，如果它是正数，则进入循环。如何更新事件呢？读入下一个数值即可，算法如下。

Read a value	初始化事件
While (value >= 0)	测试事件
...	循环主体
Read a value	更新事件
...	循环后的语句

下面我们来编写一个算法，读入数值，求正数值的和，直到读入了10个正数为止，忽略0和负数。这个例子中的事件是什么？是读入的正数值的个数和对它们求和。这意味着必须记录读入的正数的个数，我们把它叫做posCount。如何初始化事件呢？把posCount设置为0。测试posCount和10的关系，当posCount达到11时，退出循环。如何更新事件呢？每当读入一个正数时，就把posCount加1。

```
Set sum to 0                        把sum初始化为0
Set posCount to 0                   初始化事件
While (posCount <= 10)              测试事件
  Read a value
  If (value > 0)                    测试看是否应该更新事件了
    Set posCount to posCount + 1    更新事件
    Set sum to sum + value          把value加到sum上
...                                 循环之后的语句
```

许多语言还有另外两种循环结构。我们说过，while结构是预先测试循环。另一种循环结构的测试发生在循环结尾。这样的循环叫做*后测试循环*。后测试循环至少执行循环主体一次。虽然任何循环都能用while语句实现，但有时你知道循环必定要执行一次，这时还是采用后测试循环比较合适（后测试循环通常又叫做repeat循环）。还有一种结构是为计数控制的循环设计的，它的初始化、测试和增量操作都包含在循环结构自身中。这样的循环通常叫做for循环。

子程序语句

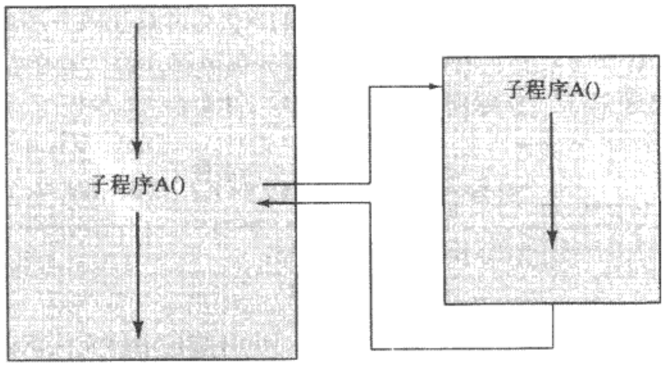
在第6章中编写算法时，我们先给每一层中的任务一个名字，然后在下一层展开这个任务。这种思想同样适用于程序设计语言。可以先给每块代码一个名字，在程序的另一部分用这个名字作为语句。当遇到这个名字后，程序这部分的处理进程将暂停，转而执行名字对应的代码。执行完指定的代码后，处理将继续执行名字之后的语句。代码名出现的位置叫做*调用部件*。

子程序有两种基本形式，一种是只执行特定任务的有名代码，一种是不仅执行任务，还返回给调用部件一个值。第一种形式的子程序在调用部件中用作语句。第二种则用作表达式，返回的值被用来评估表达式。这些子程序有很多不同的叫法。FORTRAN称它们为子程序和函数。Ada称它们为过程和函数。C++称第一种子程序为空函数，称第二种为返值函数。Java把这两种形式的子程序都叫做方法。无论怎么称呼子程序，它们都是抽象强有力的工具。使用有名子程序，用户只会看到任务完成了，不必费心了解任务的实现细节。如图8-5所示。

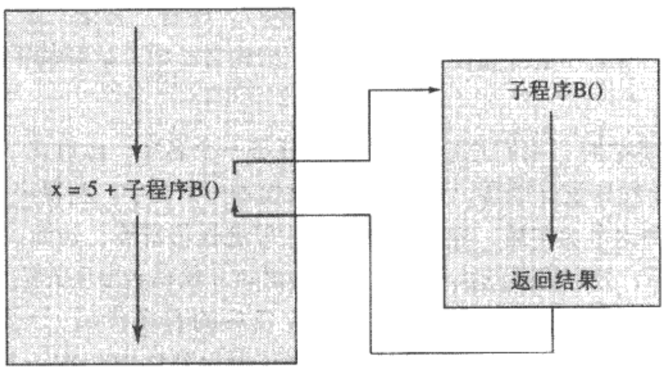
许多子程序都是高级语言或语言附带的库的一部分。例如，数学问题通常需要计算三角函数。大多数高级语言都有计算这些值的子程序。当一个程序需要计算某个三角函数时，程序员只需要查找计算这个函数的子程序名，然后调用这个子程序即可。

参数传递 有时，调用部件需要给子程序提供处理过程中需要的信息。高级语言使用的通信方法叫做**形参列表**。所谓**形参列表**，就是子程序要使用的标识符和它们的类型的列表，其中每个标识符和它们的类型放置在子程序名后的括号中。由于子程序是在被调用之前定义的，所以它不知道调用部件会传递什么样的变量。为了解决这个问题，在子程序名后面的括号中声明了一个变量名和与其相关的类型的列表。这些标识符称为**形参**。当子程序被调用时，调用部件将列出子程序名，并在其后的括号中列出一系列标识符。这些标识符叫做**实参**。实参表示的是调用部件中的真正变量。

> **形参列表**（parameter list）：程序中两部分之间的通信机制。
> **形参**（parameters）：列在子程序名后的括号中的标识符。
> **实参**（arguments）：子程序调用中列在括号中的标识符。



a) 子程序A执行完它的任务后，调用部件将继续执行下一个语句



b) 子程序B执行完它的任务后，返回一个值，这个值将被加到5上，然后存储到x中

图8-5　子程序的控制流

可以把形参看作子程序中使用的虚拟标识符。当子程序被调用时，调用部件会把真正的标识符的名字发送给子程序。子程序中的动作则是用形参定义的。当动作执行时，实参将逐个代替形参。执行替换操作的方式有几种，不过最常见的是根据位置进行替换。第一个实参代替第一个形参，第二个实参代替第二个形参，如此进行下去。

替换机制的操作有点像使用留言板。当子程序被调用时，它将得到一个实参列表（就像把实参列表写在了子程序的留言板上）。实参将告诉子程序在哪里可以找到它要用的值。子程序是通过形参在留言板上的相对位置访问它的。也就是说，子程序将在留言板的第一个位置查看它的第一个形参，在第二个位置查看第二个形参。如图8-6所示。

调用子程序时传递的实参数必须与子程序定义中的形参数相同，形参和实参在位置和数据类型上都要匹配。由于实参和形参是根据位置匹配的，所以它们的名字不必一致。当需要多次调用一个子程序，而每次调用的实参又不同时，这个属性非常有用。以这种方式传递的形参通常叫做*位置形参*。

值参和引用参数　传递参数的基本方式有两种，即通过值传递或通过引用（或地址）传递。如果一个形参是**值参**，调用部件将把实参的一个*副本*传递给子程序。如果一个形参是**引用参数**，调用部件将把实参的*地址*传递给子程序。这意味着如果传递给子程序的是实参，那么子程序不能改变它的内容，子程序修改的只是实参的副本，而原始变量并未改变。相反地，子程序可以

改变传递给引用参数的任何实参，因为子程序操作的是真正的变量，而不是它的副本。

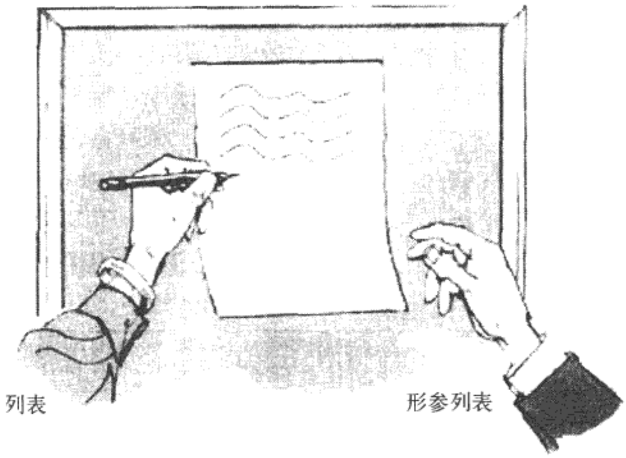


图8-6　传递参数

> **值参**（value parameter）：由调用部件传入实参的副本（写在留言板上）的形参。
> **引用参数**（reference parameter）：由调用部件传入实参的地址（写在留言板上）的形参。

可以这样看值参和引用参数，要访问一个引用参数，子程序必须访问留言板上列出的地址中的内容。要访问一个值参，子程序只需要访问留言板自身的内容即可。显然，调用部件和子程序都必须知道哪些形参/实参是通过值传递的，哪些是通过引用传递的。并非所有高级语言都支持这两种类型的参数，但支持它们的语言都有标示值参和引用参数的语法。图8-7说明了值参和引用参数之间的区别。

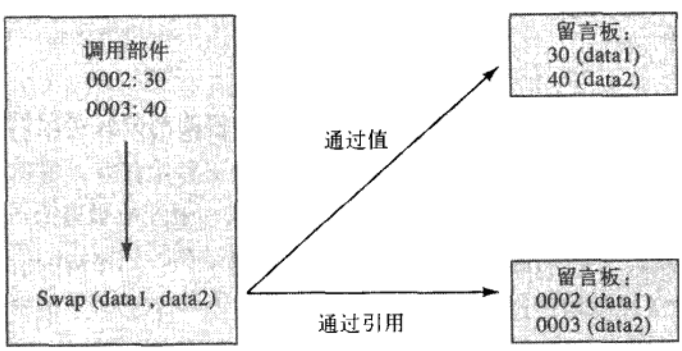


图8-7　值参和引用参数的区别

下表展示了VB.NET和C++如何定义没有返回值的子程序。这个子程序有两个整数型的值参和一个实数型的引用参数。这些例子只是为了让你体会高级语言的语法多样性，不是为了让你掌握如何用这些语言编写这一结构。C++中使用的和号（&）不是排印错误，它说明three是一个引用参数。

语　言	子程序声明
VB.NET	`Public Sub Example(ByVal one As Integer,`

（续）

语　言	子程序声明
VB.NET	`ByVal two As Integer,` `ByRef three As Single)` `...` `End Sub`
C++	`void Example(int one; int two; float& three)` `{` `...` `}`

我们没有给出Java的示例，因为Java处理内存的方式与其他三种语言有很大不同，它只有值参。没有给出Ada的示例，是因为它的编译器会自由地为既定的硬件或字节码虚拟机选择最有效的参数传递机制。

在结束子程序的介绍前，我们再看一个例子，来说明值参和引用参数之间的区别。我们来编写一个算法，交换两个内存单元data1和data2的内容。听起来非常简单，只需要把data1的内容存入data2，再把data2的内容存入data1即可。是这样的吗？不是，如果这样做，最后得到的是两个变量存放的都是data1的原始值。我们需要一个中间变量local，在把data1复制到data2之前，先把data2的内容保存在local中。中间变量只是暂时需要的。我们把子程序名和它的形参放在算法的头部。

```
Swap (Integer item1, Integer item2)
-----------------------------------
Integer temp            声明局部变量
Set temp to item2
Set item2 to item1
Set item1 to temp
```

现在调用部件（程序中需要交换两个内存单元的内容的部分）可以调用Swap子程序了，形参是data1和data2。

```
Swap (data1, data2)
```

假设data1存储在内存单元0002中，data2存储在0003中，它们存放的值分别是30和40。图8-7展示了通过值和通过引用传递参数时留言板的内容。当一个形参是值参时，子程序知道要操作的是留言板上的值。当一个形参是引用参数时，子程序则知道要操作的是留言板的地址中的内容。Swap子程序的形参应该是值参还是引用参数呢？

递归

当子程序调用自身时，这种调用称为递归调用。所谓**递归**，就是子程序调用自身的能力，是另一种循环控制结构。与其使用循环语句执行一个程序段，不如使用选择语句确定是再调用这个子程序，还是终止进程。

> **递归**（recursion）：子程序调用自身的能力。

每个递归都有两种情况，即*基本情况*和*一般情况*。基本情况是答案已知的情况，一般情

况则是调用自身来解决问题的更小版本的解决方案。由于一般情况解决的是原始问题越来越小的版本，所以程序最终将达到基本情况，由于这种情况的答案是已知的，所以递归结束。

与每个递归问题相关的是如何衡量问题的大小。每次递归调用，问题都应该减小。所有递归解决方案的第一步都是确定*尺寸系数*。如果问题涉及的是数值，尺寸系数可能就是数值本身。例如，经典的递归问题是阶乘问题。数的阶乘的定义是这个数乘以0和它之间的所有数的乘积，即$N!=N*(N-1)!$。0的阶乘是1。尺寸系数即要计算阶乘的数。基本情况是Factorial(0) = 1，一般情况是Factorial(N) = N*Factorial($N-1$)。用if语句可以判断N是等于0（基本情况）还是大于0（一般情况）。显然每次调用N都会减小，所以一定能够达到基本情况。

如果实参是负数会出现什么情况？子程序将不断地调用自身，直到运行时间支持系统耗尽了内存为止。这种情况叫做*无限递归*，与无限循环相同。

递归是非常强大的高级工具。但并非所有问题都能用递归轻易解决，即使有明显递归解决方案的问题，都未必应该用递归方式解决。不过，适用递归解决方案的问题很多。如果一个问题陈述逻辑上分为两种情况（即基本情况和一般情况），那么递归就是一种可行的方案。

异步处理

你很可能是在使用图形用户界面（GUI）的时代长大的，这种界面依靠鼠标操作屏幕上的多个窗口。“点击”成了计算机输入的主要形式。事实上，许多应用程序要求在填写完信息之后点击按钮来说明输入完成了，这已经成为主要的输入形式。

在传统的流处理中，只有当遇到指令序列中的输入语句时才执行它。下面是前面介绍过的算法中的前四条语句：

```
Write "How many pairs of values are to be entered?"
Read numberOfPairs
Set numberRead to 0
while (numberRead < numberOfPairs)
...
```

这些语句是顺序执行的。首先在窗口中显示一条输出消息，然后从输入流中读入一个name，再执行while循环。流输入和输出属于顺序的程序流。

但鼠标点击并不在程序序列中。用户可以在程序执行中的任何时刻点击鼠标。当鼠标点击发生时，程序要能够识别它，然后处理点击，再继续执行下面的操作。这种处理类型叫做**异步**，意思是“不在同一时间”。任何时刻都可以点击鼠标，这不与任何指令同步。

异步处理又叫做*事件驱动*的处理。这种处理由程序指令序列之外发生的事件控制。

异步处理在Java和VB.NET中很常用，但在其他语言中则不那么常用。不过Ada在嵌入式系统（如Flight Management Systems）中大量使用了异步处理，这些系统中的事件包括改变驾驶员座舱的按钮和开关的状态、传输机载传感器的数据和接收地面导航中心的信号等。

> **异步**（asynchronous）：不与计算机中的其他操作同时发生；换句话说，即与计算机的动作不同步。

嵌套逻辑

在任何控制结构中，要执行或要跳过的语句既可以是简单语句，也可以是块（复合语句）。

关于这些语句是什么，没有任何限制。也就是说，要跳过或重复的语句可以包含控制结构。选择语句可以嵌套在循环结构中；循环结构也可以嵌套在选择语句中。选择和循环语句可以嵌套在子程序中，子程序调用也可以嵌套在循环或选择结构中。采用计算一个文件中的正数的和的算法作为示例：

```
Set sum to 0                          把sum初始化为0
Set posCount to 0                     初始化事件
While (posCount <= 10)                测试事件
   Read a value
   If (value > 0)                     测试看是否应该更新事件了
      Set posCount to posCount + 1    更新事件
      Set sum to sum + value          把value加到sum上
...                                   循环之后的语句
```

选择控制结构嵌套在循环控制结构中。如果想计算一年中的每周的降水量，并且输出这些值，可以采用下面的嵌套循环语句：

```
Set weekCount to 1
While (weekCount<= 52)
    Set weekSum to 0
    Set dayCount to 1
    While (dayCount <= 7)
        Read rainFall
        Set weekSum to weekSum + rainFall
        Increment dayCount
    Write "Week" + weekCount + " total: " + weekSum
    Increment weekCount
```

控制结构嵌套在控制结构中，后者又嵌套在控制结构中……理论上说来，对于嵌套结构可以嵌套多少层并没有限制。不过，如果嵌套结构变得难以理解，那么最好给嵌套任务一个名字，让它成为子程序，之后再实现它。

8.3.5 复合数据类型

前面介绍的数据类型都是原子的，唯一例外的是字符串。这一节将介绍三种把数据集合起来的机制，既可以单独访问集合中的项目，又可以整体访问集合。

记录

记录是异构项目的有名集合，可以通过名字单独访问其中的项目。[10]所谓异构，表示集合中的元素可以是各种类型的。每个项目都有一个标识符和一种类型。记录的操作包括把记录作为参数传入一个子程序和访问其中的项目。记录可以把与一个对象相关的各种项目绑定在一起。例如，我们要读入一个人的姓名、年龄和时薪，可以把这三个项目集合在一个记录中。下表展示了Ada、VB.NET和C++如何声明记录类型。

语　言	记录类型的声明
Ada	`type Name_String is String (1..10);` `type Employee_Type is`

（续）

语　言	记录类型的声明
Ada	`record` `  Name : Name_String;` `  Age  : Integer range 0..100;` `  Hourly_Wage : Float range 1.0..5000.0;` `end record;`
VB.NET	`Structure Employee` `  Dim Name As String` `  Dim Age As Integer` `  Dim HourlyWage As Single` `End Structure`
C++	`struct EmployeeType` `{` `  string name;` `  int age;` `  float hourlyWage;` `};`

虽然语法有很大不同，但它们都具有三个要素，即定义了记录的类型和名字，并定义了三个变量名和它们的数据类型。注意，Ada允许程序员声明一个数值范围。Age域被定义为整数的一个合理子集。如果程序要把这个范围之外的值存入Age，程序就会出错。Hourly_wage被定义为实数的一个合理子集。这是该语句一个非常好的安全特性。

那么如何访问记录中的域呢？首先，必须声明一个记录类型的变量，然后就可以访问这个记录变量中的域了。下表展示了声明记录变量的语法以及访问这些变量的方法。

语　言	记录变量的声明和用法
Ada	`An_Employee : Employee_Type;` `...` `  An_Employee.Name := "Sarah Gale";` `  An_Employee.Age := 32;` `  An_Employee.Hourly_Wage := 95.00;`
VB.NET	`Dim AnEmployee As EmployeeType` `...` `  AnEmployee.Name : = "Sarah Gale"` `  AnEmployee.Age  : = 32` `  AnEmployee.HourlyWage : = 95.00`
C++	`EmployeeType anEmployee;` `...` `  anEmployee.name = "Sarah Gale";` `  anEmployee.age = 32;` `  anEmployee.hourlyWage = 95.00;`

Employee_Type是一个记录类型；An_Employee是一个记录变量。An_Employee可以作为参数传递，用记录中的域名可以访问其中的每个项目。这三个例子都使用记录名加点号和项目名的方式访问记录中的域。有些语言允许记录中的项目是子程序。当下一节介绍面

向对象语言的功能性时，将介绍这一点。

数组

数组是同构项目的有名集合，可以通过项目在集合中的位置访问它们。项目在集合中的位置叫做索引。有些语言把数组中的第一个元素叫做第0个项目，有些语言则允许程序员指定项目的编址方式，也就是说，第一个元素可能叫做第*a*个项目。在声明数组时，习惯上要告诉系统这个数组中有多少个项目以及它们的数据类型。下面是声明一个具有10个整数元素的数组的语法。

语　言	数组声明
Ada	`type Index_Range is range 1..10;` `type Ten_Things is array (Index_Range) of Integer;`
VB.NET	`Dim TenThings(10) As Integer`
C++/Java	`int tenThings[10];`
Java	`int[] tenThings = new int [10];`

Ada语言允许程序员指定如何访问这10个项目。在这个例子中，`Index_Range`被定义为范围1..10，它用于定义数组。这种语句组合的结果是定义了一个具有10个项目的数组，数组中的元素由范围1..10中的索引来访问。在VB.NET、C++和Java中，数组的声明指定了数组中的元素个数，而访问元素使用的则是0..9中的值。图8-8展示了这个数组中的每个单元都存储了值的情况。

[0]	1066
[1]	1492
[2]	1668
[3]	1945
[4]	1972
[5]	1510
[6]	999
[7]	1001
[8]	21
[9]	2001

图8-8　用`0..9`访问的数组变量`tenThings`

数组变量可以作为参数传递，也可以单独访问其中的单元。如何访问数组中的单元呢？用数组名加索引的方式即可。下表列出了Ada、VB.NET、C++和Java访问数组中的第三个单元和最后一个（第十个）单元的方法。

语　言	数组访问
Ada	`Put(Item => Ten-Things(3));` `Put (Item => Ten-Things (10));`
VB.NET	`MsgBox (tenThings (2) )` `MsgBox (tenThings (9) )`
C++	`cout << TenThings[2];` `cout << TenThings[9];`
Java	`System.out.print(tenThings[2]);` `System.out.print(tenThings[9]);`

记录中的变量和数组中的变量的处理方式与其他变量完全一样，只是访问方法不同而已。在记录中，变量是通过名字访问的；在数组中，变量则是通过索引访问的。

8.4 面向对象语言的功能性

面向对象的语言有三个要素，即封装、继承和多态性。这三个要素赋予了面向对象语言可重用性，从而减少了创建和维护软件的工作。让我们详细地介绍每个要素。

8.4.1 封装

第6章介绍过一些重要的计算思想，包括信息隐蔽和抽象。信息隐蔽是隐藏模块的细节，目的是为了控制对细节的访问。抽象是复杂系统的模型，只包括对观察者来说关键的细节。我们定义了三种抽象类型，但每种类型的定义采用的都是“把……的逻辑概观和它的实现分离开”这样的结构。抽象是目标，信息隐蔽是实现这一目标的方法。

在第6章中提到过，封装是把数据和动作集合在一起，数据和动作的逻辑属性与它们的实现细节是分离的。另一种说法是**封装**是实施信息隐蔽的语言特性。它用具有正式定义的接口的独立模块把实现细节隐藏了起来。一个对象只知道自身的信息，对其他对象则一无所知。如果一个对象需要另一个对象的信息，它必须向那个对象请求信息。

用于提供封装的结构叫做类。类的概念在面向对象设计中具有主导地位，同样地，类的概念也是Java和其他面向对象语言的主要特性。遗憾的是，在设计和实现阶段都没有标准化相关的概念。在设计（问题求解）阶段，**对象**是在问题背景中具有意义的事物或实体。在实现阶段，**类**是一种语言结构，这种结构是对象的模式，为封装对象类的属性和动作提供了机制。要得到一个符合模式的对象，必须用运算符和类名**实例化**这个类，实例化的结果是返回这个类的一个实例。

从语法上来说，类像前面介绍的记录，它们都是异构复合数据类型。但记录通常被认为是被动结构，只有近年来才采用子程序作为域。而类则是主动结构，一直都把子程序用作域。操作数据域的唯一方式是通过类中定义的方法（子程序）。

> **封装**（encapsulation）：实施信息隐蔽的语言特性。
>
> **对象类或类**（问题求解阶段）（object class or class (problem-solving phase)）：属性和行为相似的一组对象的说明。
>
> **对象**（问题求解阶段）（object (problem-solving phase)）：与问题背景相关的事物或实体。
>
> **对象**（实现阶段）（object (implementation phase)）：类的一个实例。
>
> **类**（实现阶段）（class (implementation phase)）：对象的模式。
>
> **实例化**（instantiate）：创建类的对象。

默认情况下，记录中的域是随便访问的，而类中的域则是私有的，也就是说，除非一个类的某个域被标识为`public`的，否则其他类的对象都不能访问这个域。如果一个类想让其他类的对象调用自己的方法，就必须明确地声明这个方法是`public`的。

`private`和`public`叫做访问修改符，它们指定了类之外的代码是否能够访问类的域。有些语言还有其他的访问代码，能够进一步设置什么代码可以访问类的域。其他类用于修改类变量的类方法都要标记为`public`的；类变量则默认或由访问代码`private`标记为`private`的。

由于类把属性和动作组合在一起，所以为一个应用程序设计的类通常还能用于其他应用程序。例如，如果编写了一个表示时间的类，并对它进行过测试，那么在任何需要时间对象

的应用程序中都可以使用它。

8.4.2 继承

继承是面向对象语言的一种属性，即一个类可以继承另一个类的数据和方法。这种关系是一种is-a关系。超类是被继承的类，派生类是继承的类。类构成了继承的体系。在这种分层体系中，所处的层次越低，对象越专门化。下级的类会继承其父类的所有行为和数据。假设定义了一个表示人的类People，具有姓名、地址、电话号码这样的数据域。在面向对象的语言中，可以定义一个Student类，让它继承People类的所有属性，然后再添加几个数据域，存放局部地址和电话号码。People类的对象只有一个地址和电话号码，而Student类的对象则有两个，一个是从People类继承来的，一个是在Student类中定义的。People类的对象只有People类的属性和行为，而Student类的对象除了具有Student本身定义的属性和行为外，还具有People类的所有属性和行为。我们说，Student类是从People类派生来的。

有了继承机制，应用程序就可以采用已经经过测试的类，从它派生出一个具有该应用程序需要的属性的类，然后向其中添加其他必要的属性和方法。

> **继承**（inheritance）：类获取其他类的属性（数据域和方法）的机制。

8.4.3 多态性

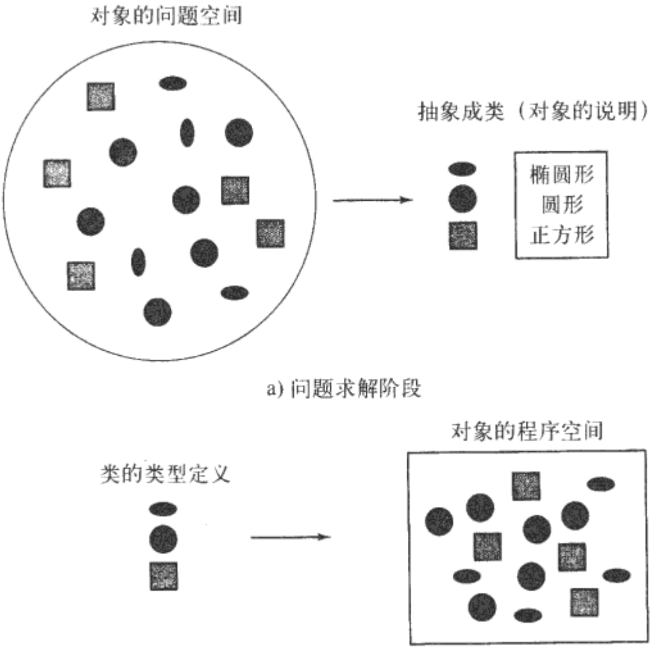
假设People类和Student类都具有一个名为printAddress的方法。在People类中，这个方法将输出People类中定义的地址。在Student类中，该方法则输出Student类中定义的地址。这两个方法名字相同，但实现不同。一种语言处理这种明显二义性的能力叫做**多态性**。语言如何知道调用部件调用的是哪个printAddress方法呢？调用部件将把类的方法应用于类的一个实例，这个对象可以确定使用的是哪个printAddress版本。

> **多态性**（polymorphism）：一种语言的继承体系结构中具有两个同名方法，且能够根据对象应用合适的方法的能力。

例如，假设jane是Person类的实例，jack是Student类的实例，jane.printAddress将调用Person类定义的方法，jack.printAddress将调用Student类定义的方法。

继承和多态性结合在一起，使程序员能够构造出在不同应用程序中可以重复使用的类的体系结构。可重用性不仅仅适用于面向对象语言，但面向对象语言的功能却使编写通用的、可重用的代码段变得更容易。

可以把问题求解阶段看作是把真实世界中的对象映射到不同的类（即对象分类的说明）中。实现阶段则根据这些分类说明（类）创建类的实例，以模拟问题中的对象。程序中对象之间的交互模拟了真实世界的问题中的对象间交互。如图8-9所示。



a) 问题求解阶段

b) 实现阶段

图8-9 问题和解决方案的映射

小结

汇编器可以把汇编语言程序翻译成机器代码。编译器可以把用高级语言编写的程序翻译成汇编语言（再被翻译成机器代码）或机器代码。解释器则不仅翻译程序中的指令，还会立即执行它们，不会输出机器代码。

高级程序设计语言的模型有四种，即流程型（命令型）、函数型、逻辑型和面向对象型。命令模型说明了要进行的处理。函数模型以函数的数学概念为基础。逻辑模型以数学逻辑为基础。面向对象模型以交互式对象的概念为基础，每个对象只负责自己的动作。

布尔表达式是关于程序状态的断言。如果断言是真的，那么布尔表达式就是true。如果断言是假的，那么布尔表达式就是false。程序用布尔表达式来判断执行哪部分代码（条件语句）或是否重复执行某段代码（循环语句）。递归是子程序调用自身的动作，它是另一种形式的循环。

程序中的每个变量都有自己的数据类型。所谓强类型化，指的是只有类型相符的值才能被存入变量。把一个值存入变量叫做给这个变量赋值（赋值语句）。

动作通常有指定的名字（子程序），当它们的名字出现在程序中的语句或表达式（过程或函数）中时，它们将被执行。子程序和调用部件之间的信息传递靠的是形参列表，即子程序名后括号中的变量和常量列表。

数据的集合可以具有名字（记录和数组）。可以通过名字（记录）或集合中的位置（数组）访问集合中的项目。

面向对象的程序设计语言具有三个要素：

- 封装：实施信息隐蔽的语言特性，用类结构实现。
- 继承：允许一个类继承另一个类的属性和行为的语言特性。
- 多态性：语言具备的消除同名方法的歧义的能力。

道德问题：开源软件的发展

如果从有专利权的软件销售商那里购买的软件出了问题，你不能修改它的源代码，然后再继续工作。源代码是制造商拥有并申请了专利的，修改、复制或转卖源代码都是违法的。开源软件则提供了另一种可选方案。开源软件允许用户以自己喜欢的任何方式修改源代码。用户不仅可以添加代码、修改代码或扩展代码，还能够复制代码，把代码给他人，甚至销售开源软件。唯一的限制是得到开源代码的用户同样具有访问、复制和销售软件的自由。这种软件特权有时叫做复制权，开源软件的支持者非常推崇这种特权。开源软件最著名的例子是Linux操作系统，它得到了Free Software Foundations General Public License的许可。

当有专利权的软件最初出现时，有团体认为它将威胁智力合作的自由度。他们认为软件是一种智力产物，因此最好将它们作为一种思想对待，欢迎任何人加入讨论，各抒己见，甚至带朋友来参与讨论。此外，如果不购买就不能使用一种软件，那么在没有把钱交给这种“思想”的主人之前，人们就不能参与它的讨论。为了响应20世纪80年代计算领域中的变化，MIT的计算机科学家们组织了自由软件基金（Free Software Foundation，FSF）来推广软件共享。波士顿的小组开发的General Public License（GPL）列出了用户共享、发布和协作开发软件产品需遵守的规则。对那些认为“free”这个名字有问题的人，FSF指出，“free”指的是“言论自由”中的“自由”，而不是“免费啤酒”中的“免费”。

那么，是什么使看来如此简单的想法充满争议呢？如果任何人都可以升级或改进一种产品，这会增加它的价值吗？开源主义的反对者们说，“不会”。Microsoft公司和其他具有软件专利权的制造商认为开源代码对它们的商业是种威胁。如果人们能够自行修补源代码，那么他们就不会支付使用专利产品所需的大量许可费，而且他们也不会购买升级产品。反对者们声称，更重要的是，开源模型有可能破坏知识产权。

开源主义的支持者们则指出了这种模型更有成本效益的一面。即使用户最初支付了软件的费用，许可协议赋予他们的自由度也不会把他们锁定在那个选择。他们可以搭配使用软件以最好地满足自己的需要。开源主义的爱好者们还指出，开源软件会趋于更可靠，发生故障的次数越来越少，IT部门和工程师修正低级问题所花费的时间也会越来越少。反对使用任何人都能够访问源代码的软件的人声称，这种做法会比使用专利软件造成更大的安全漏洞。如果航空公司、医院和市政基础设施使用这种软件，那么他们将比使用专利软件更容易受攻击。

Linux的成功给了开源组织很大的希望。它非常流行，甚至政府机关都有采用（尽管只有少数政府机关采用）。许多销售商都出售各种版本的Linux，其中包括Red Hat Linux，最著名的Linux发行商。这些实例确定了开源模型的商业可行性。有提案要求政府转用开源产品，但专利软件的制造商则致力于阻挠这一提案，而且，到目前为止，专利权和版权的法案都是支持专利软件的。这种情况是否能持续下去，只有将来才会知道。Microsoft公司提出了各种限制开源软件的方法，但迄今为止，都没有获得成功。现在，关于开源软件是造福了所有人还是危害了商业和所有权的争论仍在继续中。

练习

为练习1～14中的问题找出正确的翻译或执行系统。

A. 解释器 B. 汇编器

C. 编译器 D. 机器代码

1. 什么系统可以把高级语言翻译成机器代码？
2. 什么系统可以把Java程序翻译成字节码？
3. 什么系统可以执行字节码？
4. 什么系统可以翻译汇编语言程序？
5. 汇编器输出的是什么？
6. 什么系统以高级语言编写的程序为输入，并指导计算机执行每条语句中指定的动作？
7. 什么系统执行Java虚拟机？
8. 什么系统用于翻译ALGOL编写的程序？
9. 什么系统用于翻译APL编写的程序？
10. 什么系统用于翻译COBOL编写的程序？
11. 什么系统用于翻译FORTRAN编写的程序？
12. 什么系统用于翻译Lisp编写的程序？
13. 什么系统用于翻译SNOBOL4编写的程序？
14. 哪个翻译器运行得最慢？

为练习15～36中的语言或语言说明找出匹配的范型。

A. 命令型或流程型 B. 函数型

C. 逻辑型 D. 面向对象型

E. 具有某些面向对象特征的流程型语言

F. 具有某些流程特征的面向对象语言

15. 什么范型最确切地说明了FORTRAN语言？
16. 什么范型最确切地说明了C++语言？
17. 什么范型最确切地说明了PASCAL语言？
18. 什么范型最确切地说明了Java语言？
19. 什么范型最确切地说明了Lisp语言？
20. 什么范型最确切地说明了BASIC语言？
21. 什么范型最确切地说明了PROLOG语言？
22. 什么范型最确切地说明了SIMULA语言？
23. 什么范型最确切地说明了ALGOL语言？
24. 什么范型最确切地说明了ML语言？
25. 什么范型最确切地说明了Scheme语言？
26. 什么范型最确切地说明了Ada语言？
27. 什么范型最确切地说明了C语言？
28. 什么范型最确切地说明了Smalltalk语言？
29. 计算软件史上占统治地位的语言出自哪种范型？
30. 日本选用哪种范型作为第五代计算机使用的范型？
31. 哪种范型允许程序员用对象的分层体系表示算法？
32. 哪种范型允许程序员用任务的分层体系表示算法？
33. 哪种范型允许程序员用数学函数表示算法？
34. 哪种范型没有赋值语句？
35. 哪种范型只用递归表示循环？
36. 哪种范型没有变量？

练习37～82是问答题或简答题。

37. 汇编语言的标记是什么？
38. 请区分汇编器和编译器。
39. 请区分编译器和解释器。
40. 请比较汇编器、编译器和解释器。
41. 描述编译器提供的可移植性。
42. 描述使用字节码带来的可移植性。
43. 描述编译和运行一个Java程序的过程。
44. 在计算领域内讨论“范型”这个词的含义。
45. 列出程序设计语言的四种范型，并给出每种范型的实例。
46. 命令范型有哪些特征？
47. 函数范型有哪些特征？
48. 逻辑范型有哪些特征？
49. 面向对象程序的观点与命令式程序的观点有哪些不同？
50. 如何用程序设计语言提问？
51. 什么是布尔变量？
52. 什么是布尔表达式？
53. 给定整数变量`one`、`two`和`three`，为下列问题编写断言。
    a) `one`大于`two`和`three`吗？
    b) `one`大于`two`，但小于`three`吗？
    c) 三个变量都大于0吗？
    d) `one`小于`two`或`one`小于`three`吗？
    e) `two`大于`one`并且`three`小于`two`吗？
54. 写出布尔运算AND的运算表。
55. 写出布尔运算OR的运算表。
56. 写出布尔运算NOT的运算表。
57. 什么是数据类型？
58. 什么是强类型化？
59. 定义下列数据类型：

a) 整数　　b) 实数
c) 字符　　d) 布尔型

60. 字符串是原子数据类型吗？请证明你的答案。
61. 如果用同一个符号表示字符和字符串，如何区分单个的字符和只有一个字符的字符串？
62. 什么是声明？
63. 根据8.3.1节所示的第一个表，填写下表，展示各语言的语法标记或保留字。

语言	Ada	VB.NET	C++	Java
注释				
语句的结束标记				
赋值语句				
实数类型				
整数类型				
声明的开头				

64. Pep/7汇编语言中的.WORD和.BLOCK汇编器指示与高级语言中的声明有什么不同？
65. 请区别要翻译的指令和给翻译程序的指令。
66. 考虑下列标识符：Address、ADDRESS、AddRess、Name、NAME和NamE。
    a) 如果采用的语言是Ada，那么它们表示多少个不同的标识符？
    b) 如果采用的语言是VB.NET，那么它们表示多少个不同的标识符？
    c) 如果采用的语言是C++或Java，那么它们表示多少个不同的标识符？
67. 请解释顺序控制结构的操作。
68. 请解释if语句的控制流。
69. case语句与if语句有哪些不同？
70. 请解释while语句的控制流。
71. 什么是递归？
72. 递归如何担任循环结构？
73. 循环结构使用的是_____语句；递归结构使用的是_____语句。
74. 请解释下列语句，“子程序是抽象的强有力工具。”
75. 请说明如何用形参列表在调用部件和子程序间交流信息。
76. 请区别形参和实参。
77. 请区别值参和引用参数。
78. 记录的定义中必须具备的三个要素是什么？
79. Ada语言使用索引值的范围定义数组，而VB.NET和C++则指定数组中的元素个数。请解释它们的区别。
80. 请分析下面的三个数组声明：

```
type Index is range 21..10; -Ada
type Data_Array is array (Index) of Integer;
Data : Data_Array;                        -Ada
Dim data(11) As Integer                   'VB.NET
int data[11];                             //C++
```

这些声明定义的是同一个数组吗？请证明你的答案。

81. 请区别设计阶段的对象和实现阶段的对象的定义。
82. 请区别设计阶段的类和实现阶段的类的定义。

思考题

1. 这一章使用的示例语言的出处大相径庭。Ada是一个设计小组为美国国防部设计的。VB.NET是Microsoft公司开发的Visual Basic的最新版本。C++是贝尔实验室开发的一种系统程序设计语言，Java则是Sun Microsystems公司开发的。请推测一下每种语言的背景对这种语言产生了哪些影响。
2. 去一家电脑商店询问各种语言的编译器的价格。语言的出处对编译器的价格有影响吗？所有语言的编译器都不止一种吗？另外，这些信息让你对语言有哪些了解呢？
3. Microsoft公司为了直接与Java语言竞争，开发了一种C#语言。计算世界是瞬息万变的。在你读到这本书的时候，C#还在市场上占有一席之地吗？是不是又有关于Microsoft C#的反托拉斯宣言了？
4. 有些常见的开源软件许多人日常生活中都在使用，你能举出一些吗？
5. 你认为开源软件的质量高于或低于大型公司开发的软件吗？你如何看待开源软件的技术支持和有专利权的软件的技术支持？

第9章 抽象数据类型和算法

计算机科学有时被定义为对算法和它们在计算机中的有效实现的研究。这一章的重点是程序中模拟信息的抽象对象以及操作这些对象的算法的定义。就像拉盖式书桌用小格子组织数据一样，程序中也有表示不同类型的数据的逻辑结构。

我们首先介绍*抽象数据类型*（ADT）的概念，看看它的两种实现方法，然后从逻辑层的观点出发，讨论一些有用的ADT。也就是说，我们只分析逻辑层中这些对象的操作，并不用代码实现它们。

这一章要介绍的抽象结构之一是*列表*。我们开发了一系列操作列表项目的算法。有了这些算法，我们就可以在任何涉及列表项目的问题中使用它们。因此，最终实现算法的语言并不重要，算法分解工作可以在操作列表项目的阶段终止。

目标

学完本章之后，你应该能够：

- 定义抽象数据类型并讨论它在算法开发中的角色。
- 区别数据类型和数据结构。
- 区别基于数组的实现和链式实现。
- 区别数组和列表。
- 区别无序列表和有序列表。
- 区别选择排序和冒泡排序。
- 描述快速排序算法。
- 对列表手动应用选择排序法、冒泡排序法和快速排序法。
- 应用二分检索法。
- 区别堆栈和队列的行为。
- 把一系列项目插入二叉检索树，绘制建树的过程。
- 对一个项目序列手动模拟本章介绍的算法，证明你理解了它们。

9.1 抽象数据类型

前面几章已经用过*数据类型*这个术语。你知道什么是数据类型吗？数据类型就是一组值和能应用于这组类型的值的基本操作。**抽象数据类型**（ADT）是属性（数据和操作）明确地与实现分离的数据类型。设计的目标是通过抽象减小复杂度。如果在逻辑层定义了有用的结构和处理它们的操作，在设计中需要它们的时候，就可以随便使用它们了。

抽象数据类型（abstract data type, ADT）：属性（数据和操作）明确地与实现分离的数据类型。

为了把ADT的概念与上下文联系起来，需要看看如何观察数据。可以从应用层、逻辑层和实现层这三个方面观察数据。

应用程序（或用户）层是特定问题中的数据的视图。在面向对象的问题求解过程中，这一层将表示出特定问题中的对象。这个视图中的数据具有明确的属性和行为。

逻辑（或抽象）层是数据值（域）和处理它们的操作集合的抽象视图。在面向对象的问题求解过程中，这一层将表示出从应用层中的对象抽象出来的类。这个视图中的数据对象是具有相似的属性和行为的对象集合。用（第6章介绍的）CRC卡和定义ADT行为的责任算法可以表示这一层。

实现层明确表示出了存放数据项的结构，并用程序语言对数据的操作进行编码。这个视图用明确的数据域表示对象的属性，并用代码实现的方法表示对象的操作。这一层涉及了数据结构，即一种抽象数据类型中的复合数据域的实现。

这一章介绍的抽象数据类型是在现实世界的问题中反复出现过的。这些ADT是存储数据项的**容器**，每种ADT都具有特定的行为。称它们为容器是因为它们存在的唯一目的就是存放其他对象。

> **数据结构**（data structure）：一种抽象数据类型中的复合数据域的实现。
> **容器**（containers）：存放和操作其他对象的对象。

9.2 实现

我们说过，这一章不会谈及算法的实现，即不会去考虑代码，但我们会从逻辑层出发，介绍两种不同类型的实现，它们适用于所有程序设计语言。这两种实现分别是*基于数组的*和*链式的*。接下来的几节将介绍采用这两种实现的算法，不过仍然仅限于逻辑层。

我们在算法中使用的伪代码表达式，有些在基于数据的实现和链式实现中具有不同的意义。它们是Add item、Remove item、Get next item和More items。前两个表达式是*转换器*，因为它们将改变容器的状态。第三个表达式是*迭代器*，用它可以以一次访问一个的方式访问所有的元素。第四个是*观察者*，它将询问是否访问过所有项目了。我们来看看每个表达式在不同实现中的逻辑含义。在后面的算法中，所有步骤都是具体步骤。

9.2.1 基于数组的实现

第8章介绍过，数组是同构项目的有名集合，通过每个项目在集合中的位置可以单独访问它。项目在集合中的位置叫做*索引*。所谓**基于数组的实现**，就是用数组来存储容器中的项目的实现方法。这里并不是说数组和容器就是同一个东西，而是说可以用数组来存放容器中的项目。容器中的项目可能是无序的，也可能是有序的。如果容器中的项目是无序的，就称这个容器是*无序的*。如果容器中的项目是有序的，就称之为*有序的*。这里我们用术语*列表*表示一般类型的容器，之后将用它表示一种特定类型的容器。

> **基于数组的实现**（array-based implementation）：用数组来存储项目的容器实现方法。

图9-1展示了一个列表。这个列表由length个变量和存放这些变量的数组构成。对这个逻辑列表的逻辑操作都是从数组变量list的第0个位置开始，到第length－1个位置结束。

在无序列表中，一个项目前后的元素都与它没有什么语义关系，列表只说明了项目存储的顺序。在有序列表中，安排项目的方式则使它前后的元素与之具有语义关系。例如，分数

列表既可以是数字随机列表，又可以是根据数值排序的列表。图9-2和图9-3分别展示了无序分数列表和有序分数列表。

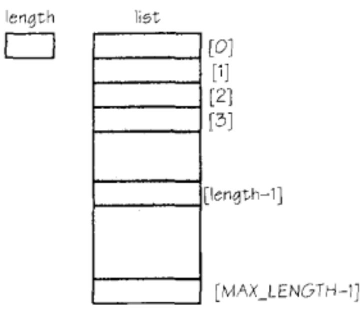


图9-1 一个列表

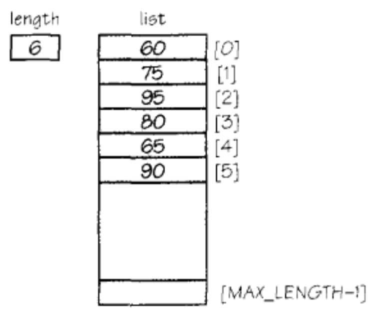


图9-2 无序的整数列表

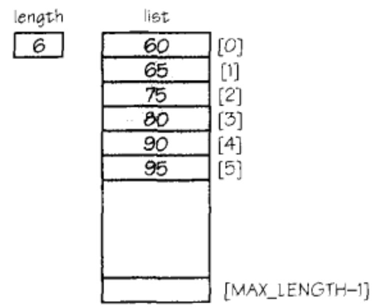


图9-3 有序的整数列表

在基于数组的实现中，访问第一个项目就是访问位置list[0]。要遍历列表中的项目，需要一个从0开始，到length为止的整数变量。记住，数组是从索引0开始，到MAX_LENGTH－1结束的；容器则是从索引0开始，到length－1结束的。忘记这一点，通常会引发程序设计中的错误。

- Add item的意思是把数组中指定索引后的元素后移一个单元，把新项目存放到索引所指的位置。
- Remove item的意思是把指定索引后的元素都前移一个单元。
- Get next item的意思是把用作索引的变量加1，再访问它索引的位置。
- More items的意思是用作索引的变量小于length－1。

9.2.2 链式实现

链式实现是以节点的概念为基础的。一个节点由两部分数据构成，一部分是用户想存入列表的项目，一部分是指向列表中的下一个节点的指针。指向列表中的第一个节点的指针保存在一个有名变量中，叫做容器（列表）的外部指针。列表中的最后一个节点的指针变量存放的是表示列表结束的符号，通常为null。图9-4展示了一个链式列表的结构图。

链式实现（linked implementation）：把项目与下一个项目的位置信息存放在一起的容器实现法。

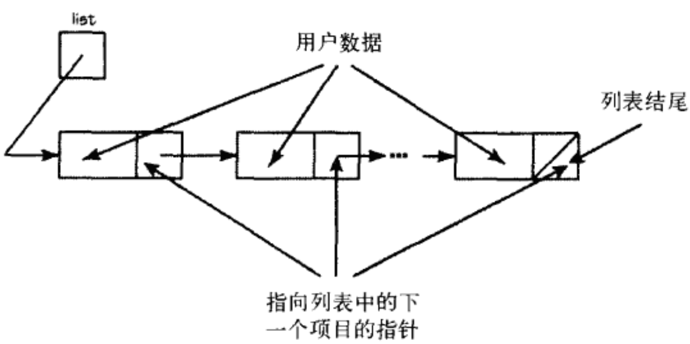


图9-4 一个链式列表的结构图

图9-5和图9-6分别展示了图9-2和图9-3中的列表。

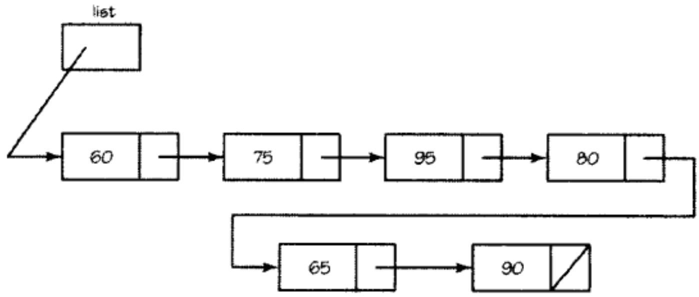


图9-5　无序链式列表

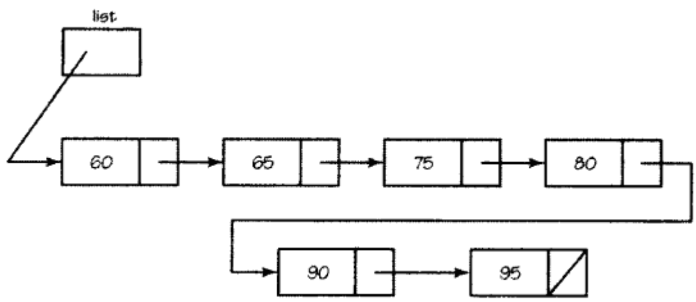


图9-6　有序链式列表

我们把用户的信息叫做节点的info部分，把指针叫做节点的next部分。在基于数组的列表中，我们使用从0到length－1的变量访问每个项目。在链式列表中，则用与节点的next部分类型相同的变量访问项目，我们称这个变量为current。current的初始值是list——列表中的第一个节点。info(current)访问的是节点中的用户数据，next(current)访问的是节点的指针部分。要转移到列表中的下一个节点，可以把current设置为next(current)。当current等于null时，访问的是最后一个节点。

Add item意思是把一个具有info的新节点插入current和next(current) 之间。如图9-7所示。

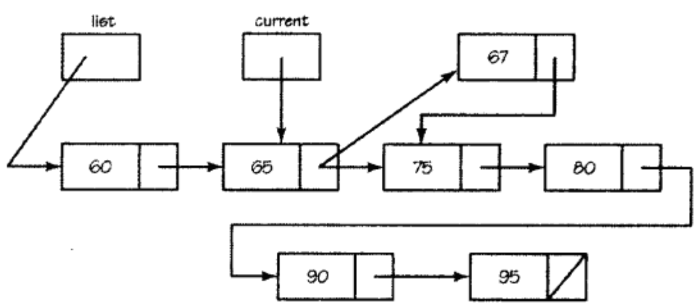


图9-7　把info为67的节点存入current之后

Remove item意思是把next(current) 处的节点删除。如图9-8所示。

Get next item意思是把current设置为next(current)。

More items意思是current不等于null。

链式列表也叫做*无限制列表*，因为节点是在运行时创建的。对节点数的唯一限制是内存的大小。在链式列表中，不必明确地记录列表中的项目个数，因为节点数是一定可以计算出的。但在基于数组的实现中，就一定要明确地保存length变量。

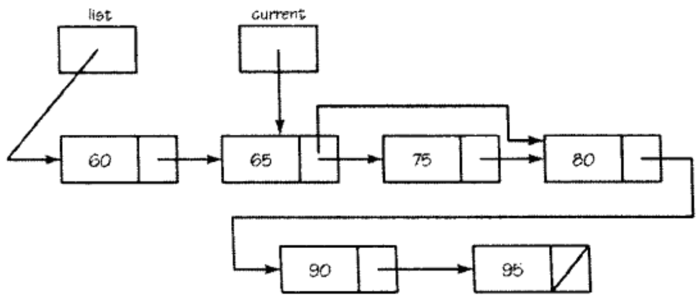


图9-8 删除节点next(current)

9.3 列表

列表出现在程序设计中，就像出现在日常生活中那么自然。我们要处理来宾列表、购物列表、分类列表以及要做的事的列表。关于列表的列表真是无穷无尽。列表有三个特征，即列表中的项目是同构的、线性的，列表是变长的。所谓线性，指的是除了第一个元素外，每个元素之前都有且只有一个元素，除了最后一个元素外，每个元素之后也有且只有一个元素。例如，如果列表中至少有三个项目，那么第二个元素就位于第一个元素之后和第三个元素之前。

9.3.1 列表的基本操作

列表具有的行为不同，它们的类型就不同。我们从最小的操作集合开始讲起。那么，操作的列表是什么呢，也就是"要做的事"的列表上有什么？是从一个空列表开始，把项目存入列表，以及从列表中去除项目。用计算机术语来说，就是给列表添加条目和从列表中删除条目。我们"要做的事"的列表是一张可以拿在手里的纸，这样就能够看到列表上的所有项目。因此，计算机模拟的列表要能够自己输出自己。我们说列表的特征之一是它具有长度，也就是说，列表知道自己存储了多少项目。综上所述，计算机化的列表要能够做到：

- 自我创建
- 插入项目
- 删除项目
- 自我输出
- 知道存放的项目个数

CRC卡是表示ADT的好方法。虽然其中的术语是面向对象的，但很容易转化成对应的函数表示法。

类名：List	超类：	子类：
责任	协作	
Initialize itself		
Insert(item)	ItemClass	
Delete(item)	ItemClass	
Print	ItemClass	
KnowLength returns integer		
⋮		

在这个CRC卡中，list类要与ItemClass类协作。虽然我们不知道ItemClass类是什么，但由于要把项目存入列表，所以列表必须与这些项目所属的对象类协作。这是一个一般列表的CRC卡。**一般数据类型**（或类）指的是规定了操作，但却没有规定要处理的对象的类型（或类）的数据类型。

> **一般数据类型**（generic data type）：规定了操作，但却没有规定要处理的对象的类型（或类）的数据类型。

算法的责任非常简明。初始化列表就是把列表长度设置为0。插入和删除项目的动作分别要把列表长度加1和减1。我们用浅色标识算法中的抽象步骤。对于上一节中提到的步骤，我们用阴影强调它们的实现方法不同。

Initialize

Set length to zero

Insert(item)

Find where the item belongs
Put the item there
Increment length

Remove(item)

Find the item
Remove the item
Decrement length

Print

While (more items)
    Get next item
    Print item

Know Length

return length

对Print操作的约束条件会产生不同类型的列表。是应该按照插入项目的顺序输出列表呢？还是应该按照项目自身的某些信息的顺序输出列表呢？按照前者输出的是无序列表，按照后者输出的则是有序列表。那么，在我们对列表项目一无所知的情况下，如何具体操作Print item呢？由于我们把这个列表看作是一般性的，所以列表中的每个项目都必须知道如何输出自己。Print item只是告诉每个项目要输出自己了。

在继续分解责任算法前，让我们来介绍一下Remove操作中的Find the item（查找项目）操作。查找一个项目，意味着该项目所属的类具有识别与之匹配的项目的方法。这不是list类的责任，而是ItemClass类的责任。这里需要明确这个假设。让我们给CRC卡添加一个假设，即ItemClass类提供了compareTo方法，并且说明了结果是什么。Java的String类和Java类库中的类都定义了compareTo方法。我们采用这种定义。此外，我们还要在有序列表中用compareTo方法进行Find where item belongs（确定项目的位置）。

<table>
<tr><td>类名：List</td><td>超类：</td><td>子类：</td></tr>
<tr><td>责任</td><td colspan="2">协作</td></tr>
<tr><td>Initialize itself</td><td colspan="2"></td></tr>
<tr><td>Insert(item)</td><td colspan="2">ItemClass</td></tr>
<tr><td>Delete(item)</td><td colspan="2">ItemClass</td></tr>
<tr><td>Print</td><td colspan="2">ItemClass</td></tr>
<tr><td>KnowLength returns integer</td><td colspan="2"></td></tr>
<tr><td>.
.
.</td><td colspan="2">假设：ItemClass提供了compareTo方法：
item1.compareTo(item2)
<0: item1<item2
0: item1=item2
>0: item1>item2</td></tr>
</table>

Find where the item belongs这个步骤有两种实现方式。如果按照插入项目的顺序（无序的）输出列表，每个项目的逻辑位置就是列表的结尾。采用基于数组的实现当然非常有效，因为插入项目的位置就是length处。但在链式实现中，只有遍历列表才能访问列表的结尾，因此，把新项目放在列表的开头更加有效。不过，采用这种实现方式，Print输出的列表就是逆序的，而不是项目插入列表的顺序。

Find where the item belongs (Unsorted)

Item belongs at length

所谓有序列表，就是按照列表项目中的某些信息的顺序输出的列表。也就是说，列表存放项目的方式必须能够让print方法中的Get an item操作得到合适的输出项目。因此，Add操作必须按照一定的顺序把项目插入列表。列表不必知道itemClass类的任何信息，compareTo方法就提供了足够的信息。

海上软件公司

一个软件企业家和一个超级邮轮的船长宣布了SeaCode的成立。这家公司具有来自世界各地的600名软件设计师，坐落在一艘沿太平洋海岸航线的豪华邮轮上，恰好在美国移民局的控制范围之外，又与美国足够近，能够参与美国的大单合约竞标。

为了说明这个过程，让我们看一个例子。假设要让数字按数字顺序排列。列表目前具有的数值如下：

23、46、75、1066、1492、2001

我们要把998插入这个列表。比较998和23，998大于23，因此比较998和下一个值46；998大于46，因此比较998和下一个值75；998大于75，因此比较998和下一个值1066；998小于1066，因此把998插入1066之前。我们从列表中的第一个项目开始，和要插入的值进行比较。只要要插入的值大于列表中的值，就移向列表中的下一个值。当找到一个列表值大于要插入的值时，这个值的位置就是要插入新项目的位置。

Find where the item belongs (Sorted)

```
Set tempItem to the first item
While (item.compareTo(tempItem) > 0)
  Set tempItem to next item
Item belongs at tempItem
```

Remove算法中的Find the item操作也可以用compareTo方法分解。在编写这个算法前，我们必须问清楚Remove的意思。可以假设要删除的项目一定存在吗？是不是只有当那个项目存在时才删除它？要删除这个项目的所有副本吗？还是只删除它的第一个副本？对这些问题的答案不同，Remove操作的含义也不同。回顾我们"要做的事"的列表，Remove操作的含义很明显——这个项目一定存在，删除它。我们根据这个假设来编写算法。但你应该注意，即使像Remove这样简单的操作，也会有多种含义，因此操作的文档必须说明算法实现的含义。Remove操作中有一个抽象步骤——Find the item。

Find the item

```
Set tempItem as first item
While (item.compareTo(tempItem) not equal to 0)
  Set tempItem to next item
```

9.3.2 其他列表操作

如何处理"要做的事"列表上的其他操作呢？在添加一个项目时，你查看过它是否已经存在于列表中了吗？你查看过列表是否是空的，是满的吗？这些操作都叫做*观察者*，因为它们要做的都是观察列表的状态。我们来分析一下确定一个项目是否已经存在于列表中的算法。这是一个布尔方法，它必须从列表头开始，用它的参数与列表中的每个项目进行比较，一旦发现一个与参数匹配的项目，就返回true。如果到达列表结尾还没有找到一个匹配的项目，则返回false。

isThere(item)

```
Set tempItem to the first item
While (more items)
  If (tempItem.compareTo(item) is equal to 0)
    return true
  Else
    Set tempItem to next item
Return false
```

这种算法叫做*顺序检索*，从列表的第一个项目开始，顺序检查每个项目。在本章结尾的练习中，将要求你完成其他观察操作的算法。

接下来的几节将分析把无序列表转化成有序列表的算法，然后介绍二分检索算法，如果我们知道列表是有序的，而且是基于数组实现的，就可以采用这种较快的检索算法。

9.4 排序

我们都知道什么是排序，抽屉中的袜子要排序，书架上的书要排序，甚至优先级都要排序。所谓排序，就是按顺序排放东西。在计算领域，把无序列表转化成有序列表是很常见的

有用操作。有很多专门介绍排序算法和有序列表的检索算法的书，它们的目的是提出更好更有效的排序算法。因为对大量元素进行排序极其耗时，所以好的排序算法非常受欢迎。有时，程序员为了得到更快的执行速度，甚至会牺牲准确性。

这一节将介绍几种完全不同的排序算法，为的是让你了解解决同一个问题有很多不同的方法。书写排序算法的语言采用的是数组符号，因为必须用索引直接访问每个要排序的项目。接下来的几节将使用compareTo方法比较两个项目。

9.4.1 选择排序

选择排序算法可能是最容易的排序算法，因为它反映了如何手动地对列表中的值排序。如果交给你一份人名列表，要求按照字母顺序对人名排序，一般的方法是：

1. 找到按字母顺序排第一的名字，把它写到另一张纸上。
2. 从原始列表中删除这个名字。
3. 继续这一循环，直到原始列表中的所有名字都被删除，写入了第二个列表，此时第二个列表就是有序的。

这个算法虽然简单，但却有缺陷，它需要两个完整列表的空间。即使不考虑内存空间，复制操作显然很费时。不过对这种手动方法稍作修改，可以免除复制空间。当从原始列表删除一个名字后，就空出了一个位置，因此不必把最小值写入第二个列表，把它与应该所在的位置处的当前值交换即可。我们用数组表示这个“手动操作列表”。来看一个例子，如图9-9所示，对具有5个元素的列表排序。由于这种算法非常简单，所以它通常是学生们学习的第一种排序方法。

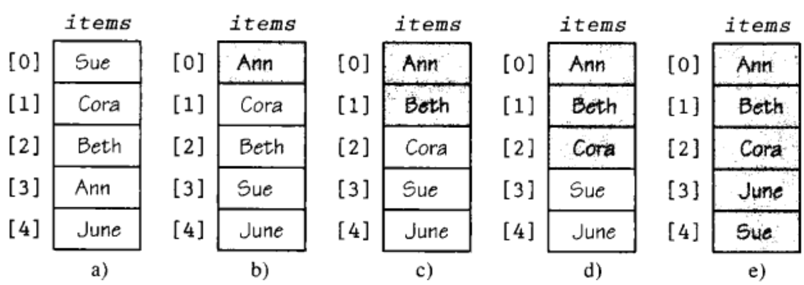


图9-9 选择排序的示例（灰色标识出了排好序的元素）

可以把这个列表看作由两部分构成，即无序部分（非灰色的部分）和有序部分（灰色部分）。每当把一个项目放到正确的位置，无序部分就缩小了，有序部分扩展了。排序开始时，所有列表项都位于无序部分；排序结束时，所有列表项都位于有序部分。

```
Selection Sort

Set current to the index of first item in the list
While (not sorted yet)
    Find the index of the smallest unsorted item
    Swap the current item with the smallest unsorted one
    Increment current to shrink unsorted part
```

这个算法中只有两个抽象步骤，即确定列表是否已经排好序了和找到最小元素的索引。从图9-9d到图9-9e，把最后两个元素添加到了列表的灰色部分。最后两个元素的操作一定是这样的，因为当最后两个最小元素之一放在了正确位置，最后一个元素一定也位于它的正确位

置了。因此，只要current小于列表的list－1，循环就会继续。

```
Not sorted yet
current < length – 1
```

如果要手动操作，那么如何在无序列表中找出按字母顺序排最小的名字呢？操作过程是看到第一个名字后，开始扫描列表，直到看到比第一个名字小的元素，记住这个更小的元素，继续扫描列表，寻找比这个元素更小的元素。这个过程总是记住迄今为止见过的最小的元素，直到扫描达到列表结尾。这个手动算法与这里要使用的算法完全相同，只是这里的算法必须记住最小元素的索引，以便与current处的项目交换。综上所述，要做的是在current到length－1这部分无序列表中寻找最小的元素。

```
Find the index of the smallest
Set indexOfSmallest to current
For index going from current + 1 to length – 1
  If (list[index].compareTo(list[indexOfSmallest]) < 0)
    Set indexOfSmallest to index
```

9.4.2 冒泡排序

冒泡排序也是一种选择排序法，只是在查找最小值时采用了不同的方法。它从列表的最后一个元素开始，比较相邻的元素对，如果下面的元素小于上面的元素，就交换这两个元素的位置（如图9-10a所示）。通过这种方法，最小的元素会“冒”到列表的顶部。每次迭代，会把未排序的最小元素放到它的正确位置，不过这同时会改变数组中其他元素的位置（如图9-10b所示）。

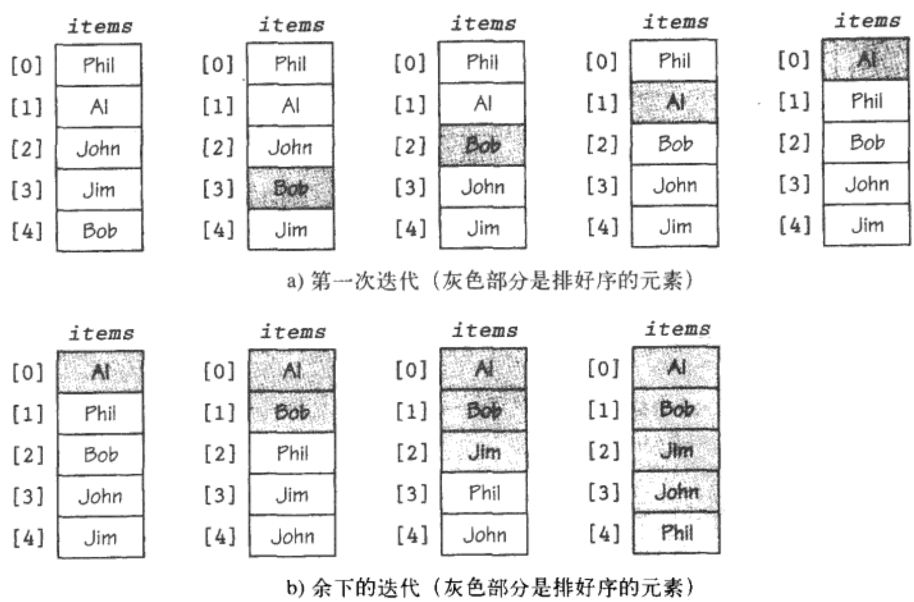


图9-10 冒泡排序的示例

在编写这个算法前，必须说明一下，冒泡排序是非常慢的排序算法。比较排序算法的方

法通常是看它们的迭代次数，而冒泡排序要对列表中除最后一个元素之外的所有元素进行一次迭代。此外，冒泡排序中还有大量的交换操作。既然冒泡排序效率这么差，为什么还要介绍它呢？因为只要对它稍加修改，就能够让它成为某些情况的最佳选择。让我们把它应用到一个已经排好序的列表上。如图9-11所示。

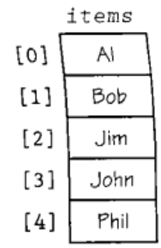


图9-11 一个已经排好序的列表

比较Phil和John，不必交换它们。再比较John和Jim，也不必交换。然后比较Jim和Bob，仍然不必交换。最后比较Bob和Al，还是不必交换。如果一次迭代都不必交换任何数据值，那么这个列表就是有序的。在进入循环之前，我们把一个布尔变量设置为false，如果在循环中发生了交换操作，才把它设置为true。如果这次循环结束时，布尔变量仍然是false，说明这个列表是有序的，整个过程不必再继续。

比较冒泡排序法和选择排序法对一个有序列表的操作。选择排序法不能确定列表是否是有序的，因此，一定要执行整个算法。

```
Bubble Sort
Set current to index of first item in the list
Do
   Set swap to false
   "Bubble up" the smallest item in unsorted part
   Increment current to shrink the unsorted portion
While (not sorted yet AND swap)
```

```
Bubble up
For index going from length – 1 down to current + 1
   If (list[index].compareTo(list[index – 1]) < 0)
      Swap list[index] and list[index – 1]
      Set swap to true
```

由于至少需要一次迭代来确定列表是否是有序的，所以我们采用后测试循环。

9.4.3 快速排序

C. A. R. Hoare开发的快速排序算法的基本思想，是对两个小列表排序比对一个大列表排序更快更容易。它的名字来源于这种算法可以相当快地对数据元素列表排序。其基本策略是分治法。

如果给你一大堆试卷，要你根据名字对它们排序，你可能会用下面的方法。先找一个分裂值（如L）把试卷分成两堆，一堆是A-L的，一堆是M-Z的。（注意，两堆中的试卷数量不必相同。）然后再把第一堆试卷分成两堆，一堆是A-F的，一堆是G-L的。A-F这堆试卷还能再分为A-C的和D-F的。分解过程将持续下去，直到每一堆足够小，能够轻易地手动排序为止。然后对M-Z的试卷应用同样的过程。

最后，把所有排好序的小试卷堆叠放在一起，就可以得到有序的试卷集合了。如图9-12所示。

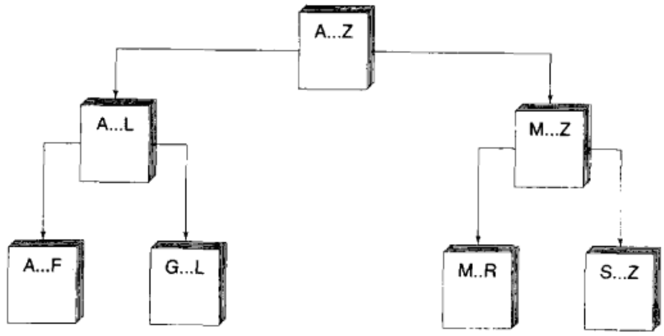
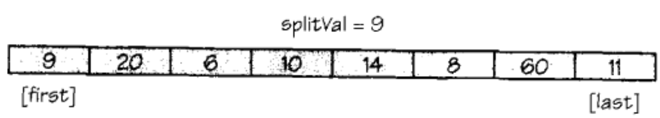


图9-12 用快速排序算法对一个列表排序

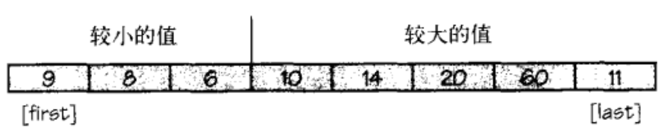
这种策略的基础是递归，每次对一堆试卷排序，都要把它分成两小堆（较小的情况），然后分别对每一小堆试卷应用同样的方法。这一过程将持续到不必再分一小堆试卷（基本情况）为止。Quicksort算法的形参列表反映出了当前正在处理的部分列表。

```
Quicksort
If (there is more than one item in list[first]..list[last])
Select splitVal
Split the list so that
   list[first]..list[splitPoint-1] <= splitVal
   list[splitPoint] = splitVal
   list[splitPoint+1]..list[last] > splitVal
Quicksort the left half
Quicksort the right half
```

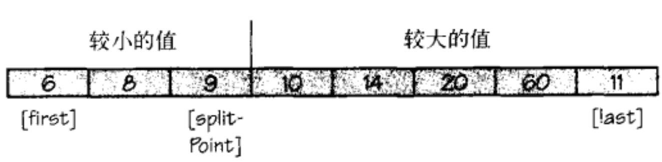
如何选择splitVal呢？一个简单的方法是用list[first]作为分裂值。我们来看一个用list[first]作为splitVal的例子。



调用Split后，所有小于等于splitVal的值都将位于列表左边，所有大于splitVal的值将位于列表右边。



splitPoint是最后一个小于等于splitVal的项目的索引。注意，只有当分裂过程完成后才会知道splitPoint的值。然后交换splitVal (list[first])和list[splitPoint]的值。



对Quicksort的递归调用使用splitPoint减小了一般情况下问题的大小。

Quicksort(first, splitPoint – 1) 将对列表的“左半边”排序，Quicksort(splitPoint+1, last) 将对列表的“右半边”排序（这里半边并不是指两边大小相同）。splitVal已经处于它的正确位置list[splitPoint]了。

那么基本情况是什么呢？当要检测的片段只有一个项目时，就不必再继续了。

必须找一种方法，把所有小于等于splitVal的元素放在列表的一边，把大于splitVal的元素放在列表的另一边。我们用一对指针，从列表的两头向中间移动，找出位置不对的元素。当发现一对位置错误的元素后，交换它们，然后继续向列表中间移动指针对。

Split

```
Set left to first + 1
Set right to last
Do
  Increment left until list[left] > splitVal OR left > right
  Decrement right until list[right] < splitVal OR left > right
  If (left < right)
     Swap list[left] and list[right]
While (left <= right)
Set splitPoint to right
Swap list[first] and list[right]
```

图9-13展示了该算法的一个例子。

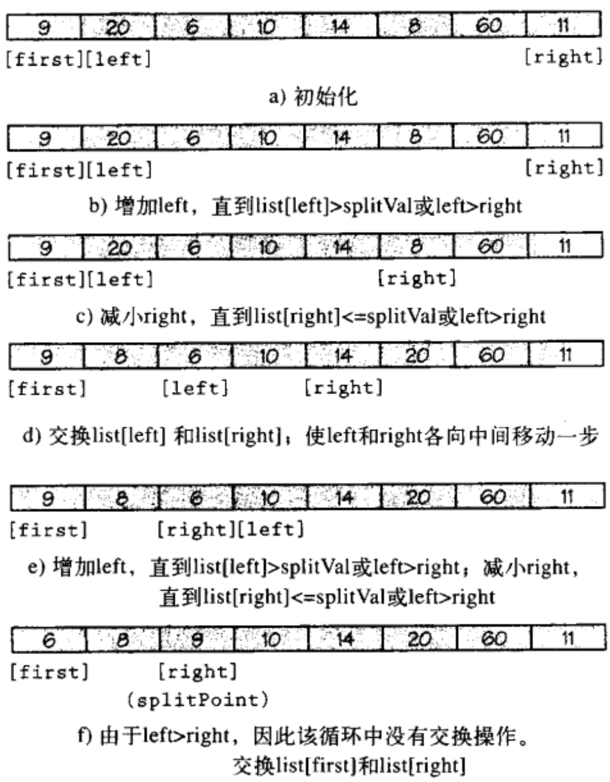
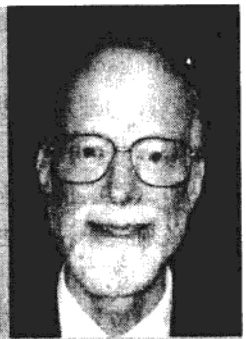


图9-13 分裂算法

Tony Hoare[1]



当Tony Hoare在John Lucas的监护下于牛津大学学习哲学（同时学习拉丁语和希腊语）时，他对计算学的兴趣被唤醒了。数学逻辑能够解释显见的数学事实的能力令他着迷。英国国民服役时期(1956—1958)，他在皇家海军学习俄语。然后他获得了统计学的证书，并偶然参与了Leslie Fox讲授的程序设计课程。1959年，他作为莫斯科国立大学的毕业生，在Kolmogorov学院从事机器翻译语言的研究。为了协助有效地查找字典中的单词，他发现了著名的快速排序算法。

1960年，他回到了英格兰，在一家小型的科学计算机制造企业Elliott Brothers担任程序员。他领导的小组（包括他后来的妻子Jill）为程序设计语言Algol 60设计并交付了第一个商业编译器。他把这个项目的成功归功于采用Algol自身作为编译器的设计语言，不过这种实现采用的是十进制的机器码。他被提升为首席工程师后，开始领导一个更大的组，从事重大的项目——实现操作系统。汲取了多次失败的经验后，他成为了计算研究部门的首席科学家，负责设计未来计算机的硬件和软件体系结构。

当这家公司与它的竞争对手合并后，这些机器被销毁了。1968年，Tony尝试申请了Belfast的女王大学的计算科学教授之职。他的研究目标是要搞清楚为什么操作系统比编译器难得多，并且要看看程序设计理论和语言是否有助于解决并发问题。虽然社会动荡，但他建立了一支强大的教学和研究队伍，并发表了一系列关于用断言来证明计算机程序的正确性的论文。他明白，这是一项长期的研究，在他的职业生涯中，这项研究未必能实现工业应用。

1977年，他来到了牛津大学，负责发展由Christopher Strachey创建的程序设计研究组。在由政府倡导、业界合作和慈善捐款得来的外部基金的协助下，牛津大学目前开办了一系列计算机科学的学位课程，包括软件工程师的外部硕士学位。他在牛津大学的研究小组追求的理想是用可证明的正确性作为计算系统（包括关键的和非关键的）的精确说明、设计和开发的主动力。这项研究的著名结果包括Z说明语言和CSP并发程序设计模型。他近来的个人研究目标是统一应用于不同程序设计语言、范型和实现方法的各种理论。

在学术领域纵横了30多年，Tony与工业界有着各种各样的联系，包括担任顾问、进行教学以及协作完成研究项目。他对维持旧代码情有独钟，其中，断言扮演着重要的角色，这样并非是为了满足他程序证明的初衷，而是为了测试代码。当他达到牛津大学的退休年龄时，他获得了重返业界的机会，在剑桥的Microsoft Research担任高级研究员。他希望增加好的学术研究的工业应用的机会，鼓励学术研究员在软件工业和客户的长期利益这一领域继续探索有深度有意义的问题。

注释：上面的传记是由Tony Hoare爵士本人拟订，在他的允许下重印的。他没有提到的是1980年他由于对程序设计语言的定义和设计的重要贡献而获得了图灵奖，以及1999年他由于对计算机科学和教育的服务而获得了爵士爵位。

9.5 二分检索法

列表的顺序检索是从列表头开始，直到找到了要找的项目或没发现该项目但已经遍历了列表为止。二分检索法查找列表项的方法则完全不同，它采用的是快速排序使用的分治法。

二分检索法假设要检索的列表是有序的，其中每次比较操作可以找到要找的列表项或把列表减少一半。该算法不是从列表头开始顺序前移，而是从列表中间开始的。如果要检索的项目小于列表的中间项，那么可以知道这个项目一定不会出现在列表的后半部分，因此只需要检索列表的前半部分即可。然后再检测这部分列表的中间项（即整个列表1/4处的项目）。如果要检索的项目大于中间项，检索将在列表的后半部分继续。如果中间项等于正在检索的项目，检索将终止。每次比较操作，都会将检索范围缩小一半。当要找的项目找到了，或可能出现这个项目的列表为空了，整个过程将终止。

> **二分检索**（binary search）：在有序列表中检索项目的操作，通过比较操作排除大部分检索范围。

这个过程是递归的。当找到了要找的项目或确定该项目不存在时，整个过程结束。如果要找的项目还可能存在，则在部分列表中查找它。在快速排序算法中，表示列表有序部分的索引是它的参数，在二分检索中也是如此。

```
Boolean Binary Search (first, last)
If (first > last)
  return false
Else
  Set middle to (first + last)/ 2
  Set result to item.compareTo(list[middle])
  If ( result is equal to 0)
    return true
  Else
    If (result < 0)
      Binary Search (first, middle – 1)
    Else
      Binary Search (middle + 1, last)
```

该算法原始调用的参数first是0，last是length－1。图9-14展示了用二分法检索cat、fish和zebra的过程。

二分检索法一定比顺序检索法快吗？表9-1展示了用顺序检索和二分检索查找项目所需的平均比较次数。如果二分检索这么快，为什么我们不总是用它呢？因为为了计算中间项的索引，每个比较操作都需要更多的计算。此外，列表必须是有序的，而且必须是用数组实现的。如果列表已经排好序了，是用数组实现的，且其中的项目不超过20个，那么使用二分检索法更好。

[0]	ant
[1]	cat
[2]	chicken
[3]	cow
[4]	deer
[5]	dog
[6]	fish
[7]	goat
[8]	horse
[9]	rat
[10]	snake

检索cat

BinarySearch(0, 10)	middle: 5	cat < dog
BinarySearch(0, 4)	middle: 2	cat < chicken
BinarySearch(0, 1)	middle: 0	cat > ant
BinarySearch(1, 1)	middle: 1	cat = cat **Return: true**

检索fish

BinarySearch(0, 10)	middle: 5	fish > dog
BinarySearch(6, 10)	middle: 8	fish < horse
BinarySearch(6, 7)	middle: 6	fish = fish **Return: true**

检索zebra

BinarySearch(0, 10)	middle: 5	zebra > dog
BinarySearch(6, 10)	middle: 8	zebra > horse
BinarySearch(9, 10)	middle: 9	zebra > rat
BinarySearch(10, 10)	middle: 10	zebra > snake
BinarySearch(11, 10)		last > first **Return: false**

图9-14 二分检索的过程

表9-1 平均比较次数

长度	顺序检索	二分检索	
		十进制	二进制
10	5.5	2.9	3.3
100	50.5	5.8	6.6
1000	500.5	9.0	9.97
10 000	5000.5	12.0	13.29

9.6 栈和队列

一提到栈，人们就会想到队列，就像提到花生酱就会想到果冻，提到马就会想到马车一

样。为什么会这样呢？因为它们的行为完全不同。

9.6.1 栈

栈是抽象数据类型的一种，只能从一端访问栈中的元素，可以在第一个位置插入元素，也可以删除第一个元素。这种ADT模拟了日常生活中的很多事情。会计师称它为LIFO，即后进先出（Last In First Out）的缩写。自助餐厅的餐具架就有这种属性。我们只能取顶上的碟子。当我们取走一个碟子后，下面的碟子就出现在了顶层，以便下一个客人取碟子。杂货架上的罐头也有这样的属性。我们取走的一行中的第一个罐头正是最后一个放入这行的。

另一种描述栈的访问行为的说法是删除的总是在栈中时间最短的项目。从这个角度观察栈就更加抽象。插入操作没有任何约束；整个LIFO行为都体现在删除操作上。

把栈比作自助餐厅的餐具架，使它的插入和删除操作有了个惯用语，插入操作叫做Push（推进），删除操作叫做Pop（弹出）。我们把项目推进栈，从栈中弹出项目。栈没有长度属性，所以没有返回栈中项目个数的操作。我们需要的是确定栈是否为Empty（空）的操作，因为当栈空的时候再Pop项目会出错。

让我们看一个使用栈的例子。在为有序列表设计插入算法时，我们说过，基于数组的实现能有效地把项目插入列表的最后一个单元，而链式实现则不那么有效。在链式实现中，插入新项目的位置明显是列表的开头，不过列表将成逆序排列的。假设插入操作把项目放在了列表的开头，可以用栈来倒着输出列表。也就是说，列表是逆序排列的，我们倒着输出列表，输出的列表就是正序的。是这样吗？我们来试试看。

```
Print list
While (more items)
   Get an item
   Push item
While (NOT IsEmpty)
   Pop item
   Print item
```

如果把下列项目

90、65、80、95、75、60

输入链式实现的无序列表，每个项目都插入列表的头部，那么该列表如图9-5所示。第一个项目将被读入，推进栈；然后第二个项目将被读入，推进栈，依此类推。当最后一个项目入栈后，栈的内容如下：

```
栈顶部→ 60
        75
        95
        80
        65
栈底部→ 90
```

下面每次弹出一个项目，输出它。输出的第一个项目是60，第二个是75，依此类推。现在用栈倒着输出了原始列表。入栈的所有项目都将按照倒序出栈。

极限编程

极限编程是小的开发组在动态环境中的一种软件开发方法。所谓极限编程，是把某些著名的软件开发实践发挥到极致的意思。例如，XP程序员总是成对地编写代码，即两个程序员在一台机器上一起工作。许多实践证明，结对编程编写的软件与程序员单独工作编写软件的开销相似，甚至更小。[2]

9.6.2 队列

队列也是抽象数据类型的一种，队列中的项目从一端入，从另一端出。会计师称之为FIFO，即先进先出（First In First Out）的缩写。听起来有点像银行或超级市场的等待队列。事实上，队列就是用来模拟这种情况的。插入操作在队列的rear（尾部）进行，删除操作在对象的front（头部）进行。

另一种描述队列的访问行为的说法是删除的总是在队列中时间最长的项目。从这个角度观察队列就更加抽象。与栈一样，插入操作没有任何约束；整个FIFO行为都体现在删除操作上。遗憾的是，插入和删除操作没有标准的相关术语。Enqueue、Enque、Enq、Enter和Insert都可以表示插入操作。Dequeue、Deque、Deq、Delete和Remove都可以表示删除操作。

9.6.3 实现

栈和队列常被看作链式结构。栈只有一个外部指针，指向栈的顶部。队列需要两个外部指针，一个指向队列的头部，另一个指向它的尾部。如图9-15所示。也可以用数组实现栈和队列。

要注意，栈和队列的删除操作都没有参数，它们都知道要删除的对象是哪一个，没有其他选择。

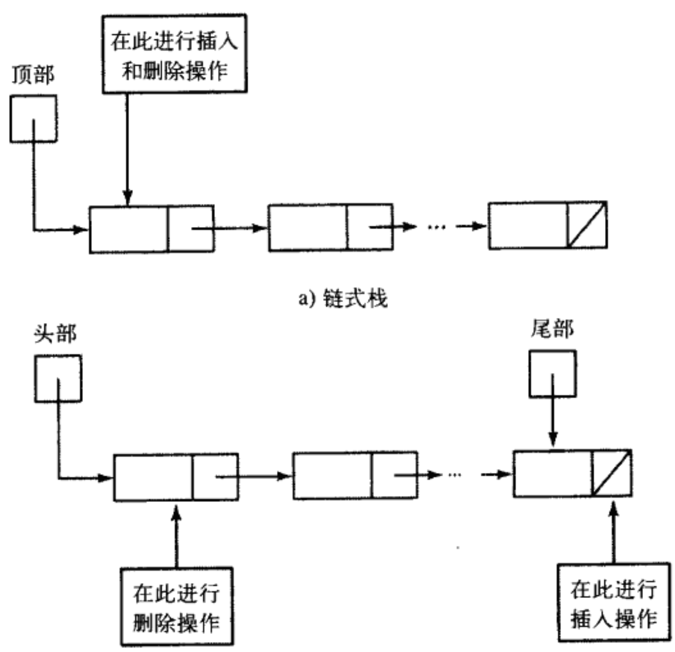


图9-15 链式结构的栈和队列

9.7 树

像列表、栈和队列这样的ADT本质上是线性的，只模拟了一种数据关系。列表中的项目一个挨着一个，栈和队列中的项目从时间上来说也是一个挨着一个的。更复杂的关系需要更复杂的结构来表示，如家族关系。如果要用程序对家族关系建模，需要一个分层体系结构，这个结构的顶层是父母，接下来的一层是儿女，再下面一层是孙子和孙女，依此类推。

这种分层体系结构叫做*树*。关于树有大量的数学理论，但在计算领域，我们所说的通常是*二叉树*，即每个节点最多有两个子节点的树。

9.7.1 二叉树

树的术语与链式实现的术语一样，树中的每个位置是一个节点，存放了用户数据和关于下一个节点的位置的信息。

从计算的角度观察，**二叉树**是一种容器对象，其中每个节点可以有两个后继节点，叫做*子女*。每个子女仍然是二叉树的节点，因此也可以有两个子女，而这些子女又可以有自己的子女，依此类推，形成了树的分支结构。树的头部是一个起始节点，叫做**根**，它不是任何节点的子女。如图9-16所示。

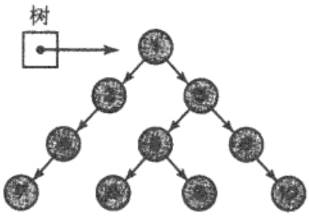


图9-16　二叉树

树的外部指针将指向它的根节点。树中的每个节点可以有0个、1个或2个子女。如果一个节点左边的子节点存在，这个子节点叫做*左子女*。例如，在图9-16中，根节点的左子女存放了值2。如果一个节点右边的子节点存在，这个子节点叫做*右子女*。在图9-16的例子中，根节点的右子女存放的值是3。如果一个节点只有一个子女，这个子女可以位于任何一边，不过它一定会位于某一边。根节点是存放值2和3的节点的父母。（以前的教科书中用术语左子、右子和父节点描述这些关系。）如果一个节点没有子女，这个节点叫做*树叶*。例如，存放值7、8、9和10的节点就是**叶节点**。

> **二叉树**（binary tree）：具有唯一起始节点（根节点）的容器对象，其中每个节点可以有两个子女节点，根节点和每个节点之间都有且只有一条路径。
>
> **叶节点**（leaf node）：没有子女的树节点。

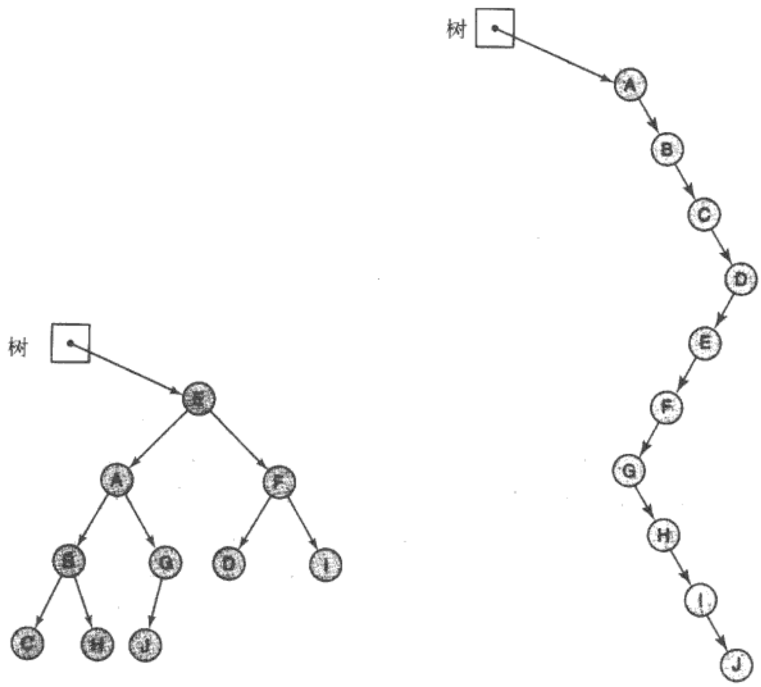
除了规定每个节点至多有两个子女外，二叉树的定义还说明了根节点和每个节点之间有且只有一条路径。这就是说，除了根节点外，每个节点都只有一个父母节点。

根节点的每个子女本身又是一个小二叉树或子树的根。在图9-16中，根节点的左子女（值为2）是它的*左子树*的根，根节点的右子女（值为3）是它的*右子树*的根。事实上，树中的每个节点都可以被看作一个子树的根。根节点的值为2的子树还包括值为4和7的节点，这两个节点是值为2的节点的子孙。值为3的节点的子孙是值为5、6、8、9和10的节点。如果一个节点是另一个节点的父母或者是另一个节点先辈的父母，那么前者是后者的*先辈*（不错，这是个递归定义）。在图9-16中，值为9的节点的先辈是值为5、3和1的节点。显然，根节点是树中其他所有节点的先辈。

节点的*层数*指的是它和根之间的距离。如果说根是第0层，那么值为2和3的节点就是第1层节点，值为4、5和6的节点是第2层节点，值为7、8、9和10的节点是第3层节点。

树的最大层数决定了它的高度。由于我们说的是二叉树，所以第N层中的最大节点数是2^N。不过，通常每层中的节点都不满。例如，在图9-16中，第2层可以存放4个节点，但因为第1层中值为2的节点只有一个子女，所以第2层只有3个节点。第3层可以存放8个节点，但例子中却只有4个。高度为N的树中的最大节点数是$2^{N+1}-1$，因此图9-16中的例子最多可以有15个节点，而实际上它只有10个。

用这个树中的10节点可以造出许多形态不同的二叉树。图9-17展示了它的两种变体。显然，具有N个节点的二叉树最多可以有N层。那么最小层数是多少呢？如果给每个节点两个子女，那么把所有节点都填入树中，这个树应该有$\log_2 N+1$层（如图9-17a所示）。你可以自己绘制一个具有8 $[\log_2(8)=3]$个节点的“完全”树，然后绘制一个有16 $[\log_2(16)=4]$个节点的“完全”树。如果有7个、12个或者18个节点又是什么情况呢？



a) 一个4层二叉树　　　　b) 一个10层二叉树

图9-17　二叉树的两种变体

恐怖分子探测软件

社会网络用数学的一个分支——图论对人们之间的交互方式进行了建模。图论把人映射为节点，把人与人之间的关系映射为连接。当前，一些研究人员利用这种方法构建了恐怖分子网络的软件模型。例如，他们在尝试判断恐怖分子网络中的命令链是否被某个行动破坏了。给这个软件输入恐怖分子网络中已经被逮捕的成员的数量，它就能估计出这个网络已经被破坏的可能性。这种估计可能比人类判断的要准确。

9.7.2　二叉检索树

*二叉检索树*具有二叉树的形状属性，也就是说，二叉检索树中的节点可以具有0个、1个或2个子女。此外，二叉检索树还具有语义属性，即任何节点的值都要大于它的左子树中的所

有节点的值，并且要小于它的右子树中的所有节点的值。如图9-18所示。

这一章前面介绍过列表的二分检索法。我们指出过，二分检索法不能应用于采用链式实现的列表。二叉检索树给列表的链式实现提供了灵活性，可以提高二分检索的速度。

在二叉检索树中检索

让我们在图9-18所示的树中检索值18。首先比较18和根节点的值15。18大于15，因此可以知道，如果18在这个树中，那么它一定在根的右子树中。注意这种检索法与线性结构的二分检索法之间的相似性，通过一次比较操作，就排除了很大一部分数据。

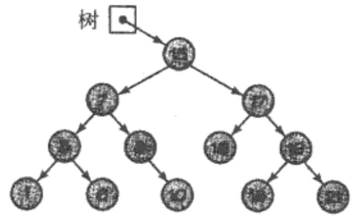
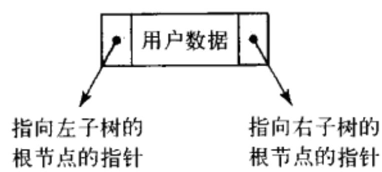


图9-18　二叉检索树

接下来比较18和右子树的根的值17。18大于17，从而可以知道，如果18在这个树中，那么它一定在根的右子树中。比较18和右子树的根的值19。18小于19，从而可以知道，如果18在这个树中，那么它一定在根的左子树中。比较18和左子树的根的值18，这样就找到了匹配的值。

下面来看看要查找的值不在树中的情况。让我们检索值4。首先比较4和15。4小于15，因此，如果4在这个树中，它一定在根的左子树中。比较4和左子树的根的值7。4小于7，因此，如果4在这个树中，它一定在7的左子树中。比较4和5。4小于5，因此，如果4在这个树中，它一定在5的左子树中。比较4和1。4大于1，因此，如果4在这个树中，它一定在1的右子树中。但1的右子树是空的，从而可知4不在这个树中。

在链式列表中，每个节点都包括一个info部分，用于存放用户数据，还包括一个指针，指向列表中的下一个节点。如果要实现一个二叉树，其中的节点就要包括三个部分，即用户数据、指向左子树的指针和指向右子树的指针。



如果current指向一个节点，那么info(current) 引用的就是这个节点中的用户数据，left(current) 指向的是current的左子树的根节点，right(current) 指向的是current的右子树的根节点。如果一个指针是null，那么这个子树就是空的。有了这些符号，就可以编写检索算法了。我们从树的根节点开始，沿着根的后继子树前进，直到找到了要找的项目或发现一个空子树为止。该算法的参数是要检索的项目和树（子树）的根节点。

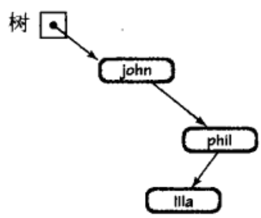
```
Boolean IsThere(current, item)

If (current is null)
   return false
Else
   Set result to item.compareTo(info(current))
   If (result is equal to 0)
      return true
   Else
      If (result < 0)
         IsThere(item.left(current))
      Else
         IsThere(item.right(current))
```

每次比较操作，不是找到了检索的项目，就是把树减小了一半。当然，说一半并不精确。如图9-17所示，二叉树的形状并不总是均衡的。显然，二叉检索树的检索算法的有效性与树的形状有直接关系。树的形状是如何形成的呢？树的形状是由项目插入树的顺序决定的。让我们来建一个二叉检索树。

构造二叉检索树

我们刚使用过的检索算法为如何构造二叉检索树提供了线索。如果在检索路径中没有找到要找的项目，那么最后达到的就是这个项目应该在的位置。下面用字符串john、phil、lila、kate、becca、judy、june、mari、jim和sarah构造一个二叉检索树。



因为john是第一个插入的值，所以它是根节点。第二个值phil大于john，因此它将成为右子树的根节点。lila大于john，但小于phil，因此它将成为phil的左子树的根节点。此时该树如右图所示。

kate大于john，小于phil和lila，因此kate将成为lila的左子树的根节点。becca小于john，因此它将成为john的左子树的根节点。judy大于john，小于phil、lila和kate，因此judy将成为kate的左子树的根节点。june的路径与judy一样。june大于judy，因此它将成为judy的右子树的根节点。mari将成为lila的右子树的根；jim将成为becca的右子树的根；sarah将成为phil的右子树的根。整个树如图9-19所示。

```
Insert(current, item)

If (tree is null)
   Put item in tree
Else
   If (item.compareTo(info(current)) < 0)
      Insert (item, left(current))
   Else
      Insert (item, right(current))
```

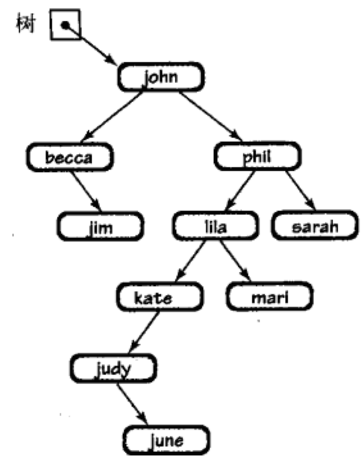


图9-19 字符串的二叉检索树

图9-20展示了在图9-19的树中插入nell的操作过程。我们用括号括起节点info部分的内容表示指针，指向以该值作为根的子树。

调用insert操作	第一个if语句	第二个if语句	动作/调用
Insert((john), nell)	(john)! = null	nell>john	插入右子树
Insert((phil), nell)	(phil)! = null	nell<phil	插入左子树
Insert((lila), nell)	(lila)! = null	nell>lila	插入右子树
Insert((mari), nell)	(mari)! = null	nell>mari	插入右子树
Insert((null), nell)	null = null		把nell存为(mari)的右子树的根

图9-20 在图9-19的树中插入nell的操作过程

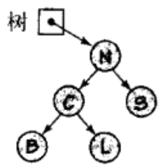
输出二叉检索树中的数据

在输出列表中的值时，我们使用了表达式Get next item，并且分别说明了如何用数组和链式实现来实现它。这个表达式的逻辑意义非常明显，即按照线性顺序获取列表中的下一个项目。那么在二叉检索树中这个表达式的意义是什么呢？就目的来说，它的意义是一样的，只是它不再线性地输出数据，让我们从树的角度来观察它。

要输出根的值，必须先输出它的左子树中的所有值，即所有比根的值小的值。输出了根的值后，还必须输出它的右子树中的所有值，即所有比根的值大的值。这样就结束了吗？那左子树和右子树中的值怎么办？如何输出它们？当然采用相同的方法。毕竟，它们都是二叉检索树。这个算法听起来很容易。这就是递归算法的妙处，简短精致（尽管有时要思考一番）。让我们用算法编写下面显示的树并跟踪它的执行过程。由于有两个递归调用，所以我们对跟踪过程中的调用进行了编号。

```
Print (tree)

If (tree is NOT null)
   Print (left(tree))
   Write info(tree)
   Print(right(tree))
```



调用编号	调用print操作	if语句	动作/调用
1	Print((N))	(N)! = null	输出左子树
2	Print((C))	(C)! = null	输出左子树
3	Print((B))	(B)! = null	输出左子树
4	Print((null))	(null) = null	返回3
3			输出B，输出右子树
5	Print((null))	(null) = null	返回2，结束调用3
2			输出C，输出右子树
6	Print((L))	(L)! = null	输出左子树
7	Print((null))	(null) = null	返回6
6			输出L，输出右子树
8	Print((null))	(null) = null	返回1，结束调用6和2
1			输出N，输出右子树
9	Print((S))	(S)! = null	输出左子树
10	Print((null))	(null) = null	返回9
9			输出S，输出右子树
11	Print((null))	(null) = null	返回原始调用，结束调用9和1

该算法按升序输出了二叉检索树中的项目。还有其他遍历树的方法，可以按照其他顺序输出树中的项目。我们将在练习中探讨这些遍历方法。

9.7.3 其他操作

现在你应该意识到了，二叉检索树其实是和列表具有同样功能的对象，它们的区别在于操作的有效性，而行为是相同的。我们没有介绍Remove算法，因为它对于本书来说太复杂了。此外，我们还忽略了length的概念。与其在构造树的时候记录其中的项目个数，不如编写一个算法，计算树中的节点数。

一个空树中有多少个节点？当然是0个。那么任意一个树有多少个节点呢？就是1加上左子树中的节点个数和右子树中的节点个数。树的定义导致了length操作的递归定义。

```
Integer Length(tree)

If (tree is null)
  return 0
Else
  return Length(left(tree)) + Length(right(tree)) + 1
```

9.7.4 图

树是表示存在层级的关系的有效方式，也就是说，一个节点至多只有一个指向它的节点（它的父母）。如果去掉这种约束，就得到了另一种数据结构——**图**。图由一组节点和连接节点的线段构成，图中的节点叫做**顶点**，图中的线段叫做**边**（或**弧**）。

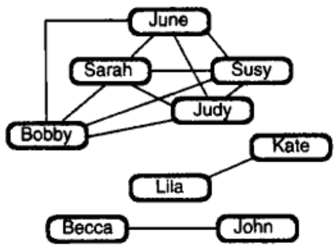
图中的顶点表示对象，边则描述了顶点之间的关系。例如，如果一个图表示地图，那么顶点就可能是城市的名字，连接顶点的边表示的就是两个城市之间的公路。由于城市之间的公路都是双向的，因此这个图中的边是无向的。这种图叫做**无向图**。但如果连接顶点的边表示从一个城市到另一个城市的航道，那么每条边的方向就很重要了。存在从Houston到Austin的航道并不意味着存在从Austin到Houston的航道。其中的边是由一个顶点指向另一个顶点的图叫做**有向图**。

> **图**（graph）：由一组节点和一组把节点连接起来的边构成的数据结构。
>
> **顶点**（vertex）：图中的节点。
>
> **边（弧）**（edge(arc)）：表示图中两个节点的连接的顶点对。
>
> **无向图**（undirected graph）：其中的边没有方向的图。
>
> **有向图**（directed graph(digraph)）：其中的边是从一个顶点指向另一个顶点（或同一个顶点）的图。

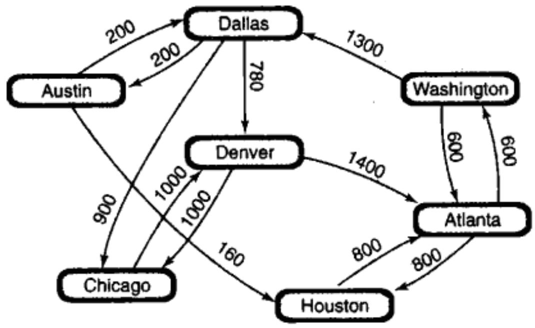
顶点表示的对象可以是人、房子、城市、课程、概念，等等。边表示对象之间的关系。例如，人和人之间是相关的，同一条街道上的房子是相关的，有向航道把城市连接了起来，课程之间也存在先决关系，一个概念则可能是由另一个概念派生的。但图9-21所示。从数学上来说，顶点是图论中未定义的概念。有关图的数学问题多种多样，不在本书的讨论范围内。但有必要指出，本章前面介绍过的栈和队列可以用来处理图。

9.8 程序设计库

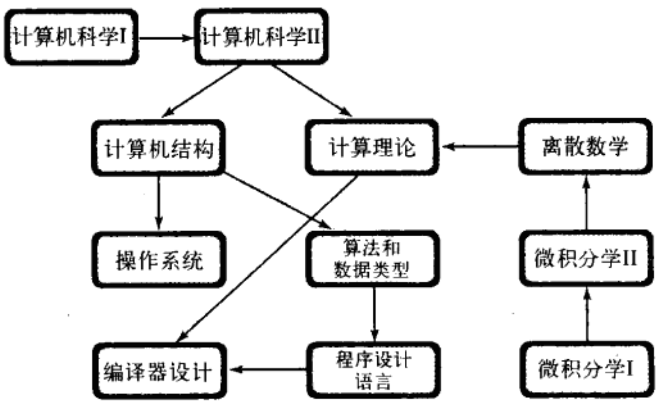
大多数现代程序设计语言都为程序员提供了类库和编好的算法。这些库程序大部分是封装了抽象数据类型的类。遗憾的是，这些库的设计者没有用经典的名字表示库中的对象，因此要找到你想用的类有点困难。但在计算领域绝对不要闭门造车，一定要检查你使用的语言的类库，看自己想模拟的行为是不是已经存在了。



a) 顶点：人 边：兄弟姐妹关系



b) 顶点：城市 边：有向航道



c) 顶点：课程 边：先决条件

图9-21 图的示例

小结

抽象数据类型（ADT）是属性（数据和操作）与实现细节分离的数据类型。容器是存储其他对象的对象。我们用ADT来描述容器对象。ADT有两种常用的实现方法，即基于数组的实现和链式实现。基于数组的容器用数组存储对象；链式实现的对象则包含指向下一个对象的指针。

列表、栈、队列、树和图都是有用的容器型ADT。每种容器都有自己特定的属性和确保这些属性的操作。所有ADT都有插入和删除项目的操作。列表和树还有查找项目的操作。

所谓排序，就是把列表中的项目按照一定的顺序排列。选择排序法、冒泡排序法和快速排序法是三种常用的排序算法。如果列表是有序的，则可以用专用的检索算法——二分检索法来查找列表中的项目。

程序设计库是为程序设计员设计的类和算法的集合。在开始编码前，程序员应该查询采用的语言的程序设计库，看看是不是已经有可用的类提供了需要的行为。

道德问题：使用计算机的恶作剧和欺诈行为

自从人们知道可以利用别人起，就存在诈骗犯、骗子和搞恶作剧的人。在计算机出现之前，这些人行骗的生活还很艰难。他们必须先找到目标，然后花费大量的时间和金钱慢慢套取对方的信任。

每次他们只能锁定有限的目标。如果使用邮件，他们就要支付纸张、信封和邮票的费用，更不要提收集姓名和地址信息需要花费的时间。

自从有了Internet，他们只要点击几下鼠标，就可以把诈骗信发给成千上万个收信人。而且，收集电子邮件地址的工作还可以由计算机自动完成。潜在的受害者人数众多。网站可以诱捕那些误入的用户，除了伪造网站内容外，他们不需再为诈骗付出任何努力。

许多人认为商业信息兜售是在线骚扰的极限。殊不知那些肆无忌惮之徒可以利用科技为所欲为。有些人甚至可以骚扰数千人。他们会发送恶作剧消息，指示收信人拨打长途电话，以便从名单上删除自己的信息。另一种恶作剧是发送电子邮件，告知收信人如何通过更改计算机设置杜绝垃圾邮件。当然，没有戒心的受害者根据这些说明进行操作只会使计算机再也无法正常运转。这种恼人的诡计层出不穷。然而，在电脑网络空间中，还有人会制造更加严重的问题。

许多罪犯使用电子邮件和Internet诈骗了上百万美元。有些电子邮件会恳请给假冒的慈善机构或贫困的人捐款。这种模式利用了人们帮助他人的愿望，其中代价最高、最著名的一例叫做尼日利亚诡计。这种诡计利用蹩脚的英语恳求仁慈的收信人提供经济援助。发信人毫无例外都自称是高级官员，他们有几百万的黑钱。如果收信人帮助逃跑的官员转移资产，他就会收到一份钱。只是这种阴谋造成的平均损失额就超过了5000美元。其他骗局还有骗收信人开立由第三者保存的账户。还有伪造拍卖，告知买主把钱存入某个账户的情况。当然，买主拍得的货物永远也不会到手，由第三者保存的账户也会从此消失了。

更严重的是从网上偷盗商业信息和密码的罪犯。他们会使用网站诱使人们相信自己正在做问卷调查，从而提供信用卡信息，只是为了证明他们已满18岁了。盗取了密码后，罪犯就可以访问整个商业记录。身份被盗用对受害者来说是巨大的打击，可能需要多年才能免受其苦。最大的威胁还是来自于那些要制造毁灭性打击的罪犯。航空公司、银行和市政基础设施无不与计算机网络相关。由计算机罪犯造成的破坏程度是难以估计的。

打击这种犯罪的难度很高。受害者可能分布在世界各地。犯罪者不仅能辨识出受害者的身份，还能确认他们的地理位置。目前，用户最好的保护方式是保持怀疑态度。如果一封电子邮件要你给所有认识的人发送一条消息，那么它很可能是一个恶作剧，无论听起来多么可信。对于想参与在线拍卖的人，在竞拍或付款之前，最好先致电Better Business Bureau，这样可以节省时间和金钱，还能避免受骗。对于任何请求都不能提供自己的信用卡号码和个人信息。随着计算机的应用更加广泛，诈骗犯、搞恶作剧的人和骗子将与之同步发展。除非找出一种可行的制止方案，否则在网上冲浪时要时刻保持警惕。

练习

为练习1～10中的操作找出匹配的操作类。

A. 构造器：创建新的容器实例

B. 转换器：改变容器的内容

C. 迭代器：允许每次访问容器中的一个项目

D. 观察者：查询容器的信息，但不改变它

1. 把一个项目放入容器。
2. 创建一个新容器。
3. 容器中有多少对象？
4. 容器是空的吗？
5. 获取容器中的下一个项目。
6. 我们已经看过容器中的所有项目了吗？
7. 删除容器中的一个项目。
8. 这个项目还在容器中吗？
9. 把容器设置为空的。
10. 容器满了吗？

为练习11～20中的实现步骤找到匹配的实现类型。

A. 基于数组的　　　B. 链式的
C. 两者兼具的　　　D. 两者都不是的

11. 通过把长度设置为0进行初始化。
12. 在一个无序容器中，把项目放在列表的开头。
13. 在一个有序容器中，把项目放在列表的结尾。
14. 要把一个项目插入一个有序列表，首先要找到正确的插入位置。
15. 通过把指针设置为null进行初始化。
16. 通过给计数器加1获取下一个项目。
17. 通过移动指针获取下一个项目。
18. 通过比较计数器和长度值确定是否还获取更多的项目。
19. 通过比较指针和null确定是否还获取更多的项目。
20. 通过检索列表查找项目。

判断练习21～37中的陈述的对错：

A. 对　　　B. 错

21. 不能对链式列表应用二分检索。
22. 不能对基于数组的列表应用线性检索。
23. 不能对基于数组的列表应用二分检索。
24. 二分检索法一定比线性检索法快。
25. 不能对无序列表应用二分检索。
26. 栈具有LIFO属性。
27. 队列具有FIFO属性。
28. 栈和队列是同一种抽象数据类型的不同版本。
29. 在链式结构中，二叉检索树最多需要$\log_2 N$次检索。
30. 在图中，顶点表示要建模的对象。
31. 冒泡排序算法要找到数组无序部分的最小项目，然后把它与当前项目交换。
32. 选择排序算法要找到数组无序部分的最小项目，然后把它与当前项目交换。
33. 冒泡排序算法要交换它发现的每个无序对。
34. 快速排序算法要交换它从数组两端遇到的所有无序对。
35. 快速排序算法总是很快。
36. 二叉检索树的形状是由插入项目的顺序决定的。
37. 图中的边表示关系。

练习38～58是问答题或简答题。

38. 抽象数据类型、数据结构和容器：
    a) 定义这些术语。
    b) 它们有哪些共同之处?
    c) 它们有什么区别?
39. 列出数据的三种视图，并描述它们。
40. 基于数组的实现和链式实现：
    a) 定义这两个术语。
    b) 它们有哪些共同之处?
    c) 它们有什么区别?
41. 绘制一个无序列表，包含下列字符串：blue、black、green、yellow、red、purple、white和violet。
    a) 采用基于无序数组的列表。
    b) 采用基于有序数组的列表。
    c) 采用无序链式列表。
    d) 采用有序链式列表。
42. 描述下列表达式在基于数组的实现中的意义。
    a) Add item　　　b) Remove the item
    c) Get next item　　　d) More items
43. 描述下列表达式在链式实现中的意义。
    a) Add item　　　b) Remove the item
    c) Get next item　　　d) More items

问题44～46要使用下面的列表。

length: 11

list

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
23	41	66	20	2	90	9	34	19	40	99

44. 在选择排序中，首先把current设置为第4个项目，请展示列表的状态。
45. 在冒泡排序中，首先把current设置为第5个项目，请展示列表的状态。
46. 在快速排序中，用list[0]作为第一次递归调用的分裂值，请展示这次递归后列表的状态。

问题47和48要使用下面的列表。

length: 11

list

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
5	7	20	33	44	46	48	99	101	102	105

47. 在顺序检索中，要找到下列值或确定它们是否在列表中，需要多少次比较操作?
    a) 4　　　b) 44　　　c) 45
    d) 105　　　e) 106
48. 在二分检索中，要找到下列值或确定它们是否在列表中，需要多少次比较操作?
    a) 4　　　b) 44　　　c) 45

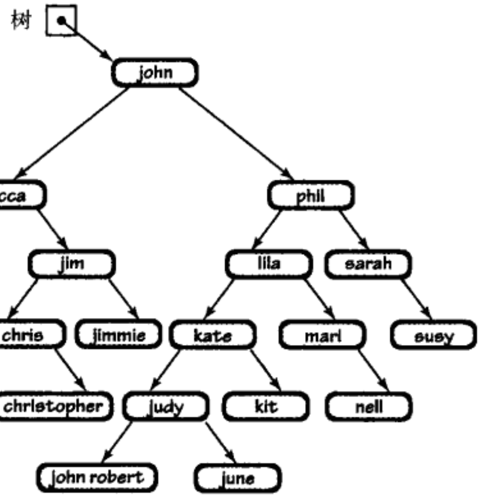
d) 105 e) 106

49. 用基于数组的实现编写Enque的算法。

50. 用链式实现编写Pop的算法。

51. 用链式实现编写Deque的算法。

练习52～57要使用下面的树。



52. 列出每个叶节点的内容。

53. 列出所有只有右子女的节点的内容。

54. 列出所有只有左子女的节点的内容。

55. 这个树的高度是多少？

56. 列出kit节点的先辈节点的内容。

57. jim节点具有多少个子孙节点。

58. 绘制按照下列顺序插入值生成的树：

25、15、7、30、26、8、2、40、35

思考题

1. 电子数据表是一种由行和列构成的表格。考虑一种ADT电子数据表。构造这种表格需要什么操作？处理表中的数据需要什么操作？

2. 链式列表、树和图都由节点和表示节点间关系的箭头（指针）构成。从能够应用于这些结构的操作这方面来比较它们。列表可以是树吗？树可以是列表吗？树可以是图吗？图可以是树吗？这些结构是如何关联起来的？

3. 你曾经经历过或进行过Internet诈骗吗？这种欺诈会表现得非常真诚吗？你能立刻辨认出它或者仔细阅读它来确定真伪吗？

4. 哪些政府机关和执法机构有助于你与身份盗用作斗争？

第五部分 操作系统层

第10章 操作系统

要理解计算机系统，必须理解管理和协调各个部件的软件。计算机的操作系统把硬件和软件紧密地结合在一起，它是其他软件依附的基础，并且允许我们编写与机器进行交互的程序。本章和下一章将探讨操作系统管理计算机资源的方法。就像交警要使通过十字路口的车流井井有条一样，操作系统要使通过计算机系统的程序流井然有序。

目标

学完本章之后，你应该能够：

- 描述操作系统的两个主要责任。
- 定义内存和进程管理。
- 解释分时操作是如何创建虚拟机假象的。
- 解释逻辑地址和物理地址之间的关系。
- 比较内存管理方法。
- 区别固定分区法和动态分区法。
- 定义和应用分区选择算法。
- 解释请求分页是如何创建虚拟机假象的。
- 解释进程生存周期的各个阶段和过渡。
- 解释各种CPU调度算法的处理。

10.1 操作系统的角色

第1章介绍过程序员角色的变迁。早在第一代软件开发的末期，程序员就分为编写工具以帮助他人的程序员和使用工具解决问题的程序员。现代软件可以分为两类，即应用软件和系统软件，反映了程序设计目的的分类。**应用软件**是为了满足特定需要——解决真实世界中的问题——而编写的。文字处理程序、游戏、库存控制系统、汽车诊断程序和导弹控制程序都是应用软件。第12～14章将讨论计算机科学的各个领域以及它们和应用软件之间的关系。

系统软件负责在基础层上管理计算机系统。它为创建和运行应用软件提供了工具和环境。系统软件通常直接与硬件交互，提供的功能比硬件自身提供的更多。

计算机的**操作系统**是系统软件的核心。操作系统负责管理计算机的资源（如内存和输入/输出设备），并提供人机交互的界面。其他系统软件则支持特定的目的，如在屏幕上绘制图像的图形软件的库。操作系统允许一个应用程序与其他系统资源进行交互。

应用软件（application software）：帮助我们解决现实世界问题的程序。
系统软件（system software）：管理计算机系统并与硬件进行交互的程序。
操作系统（operating system）：管理计算机资源并为系统交互提供界面的系统软件。

图10-1展示了操作系统在计算机系统元素中的相对位置。操作系统负责管理硬件资源。它允许应用软件直接地或通过其他系统软件访问系统资源。它提供了直接的人机交互界面。

一台计算机通常只有一个活动的操作系统，在系统运行中负责控制工作。计算机硬件是靠电线连接的，初始时永久性存储器（ROM）中存有一小组系统指令。这些指令将从二级存储器（通常是硬盘）中载入大部分系统软件。最终将载入操作系统软件的所有关键元素，执行启动程序，提供用户界面，系统就准备就绪了。这个过程叫做*引导计算机*。术语“引导”来自于“靠自己的努力振作起来”这一思想，这也正是打开计算机后它所做的事情。

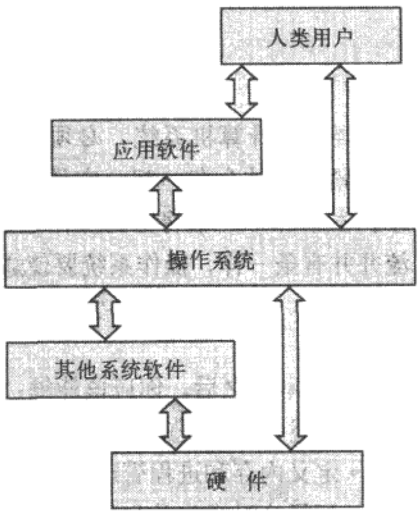


图10-1　操作系统能与计算机系统的多个元素进行交互

计算机可以具备两到三个操作系统，用户在打开计算机时可以选择使用哪个操作系统。这种配置称为*双引导*或*多引导*系统。不过，任何时候都只有一个操作系统在控制计算机。

你可能习惯于使用一种操作系统。个人计算机常用的是Microsoft Windows的各种版本（Windows 2000、Windows NT、Windows XP和Windows Vista）。这些操作系统的不同版本代表了软件的进化以及提供的服务方式和管理的不同。Mac OS是Apple Computer公司制造的计算机采用的操作系统。严格的程序员多年来都喜欢采用UNIX操作系统，最近，个人计算机流行使用的操作系统是UNIX的一个版本，叫做Linux。

任何操作系统都以自己特定的方式管理计算机资源。本章的目标不是分析操作系统间的不同之处，而是讨论它们的共同点。我们偶尔也会提到某个特定OS（操作系统）使用的方法，讨论它们独有的理念。但总的来说，我们的重点是底层的概念。

操作系统的各种角色通常都围绕着一个中心思想“良好的共享”。操作系统负责管理计算机的资源，而这些资源通常是由使用它们的程序共享的。多个并发执行的程序将共享主存，依次使用CPU，竞争使用输入/输出设备的机会。操作系统将担任现场监视器，确保每个程序都能够得到执行的机会。

谁是Blake Ross

Blake Ross从10岁就开始设计网页了。14岁时，他把修复Netscape浏览器中的bug作为一项爱好。在中学毕业前，他协助开发了开源的网络浏览器Firefox。American Online公司建立了一个非赢利性的基金，以资助Firefox的开发，2004年11月，Firefox正式发行了。在大学期间，Ross仍然在修复Firefox中的bug。

10.1.1 内存、进程和CPU管理

第5章介绍过，正在执行的程序都驻留在主存中，其中的指令以读取-解码-执行这种周期性方式被一个接一个地处理。**多道程序设计**是在主存中同时驻留多个程序的技术；这些程序为了能够执行，将竞争CPU资源。所有现代操作系统都采用多道程序设计技术，因此，操作系统必须执行**内存管理**，以明确内存中有哪些程序，以及它们驻留在内存的什么位置。

操作系统的另一个关键概念是**进程**，可以将它定义为正在执行的程序。程序只是一套静态指令，进程则是动态的实体，表示正在执行的程序。在多道程序设计系统中，可能同时具有多个活动进程。操作系统必须仔细管理这些进程。无论何时，下一条要执行的都是一条明确的指令。中间值将被计算出来。在执行过程中，进程可能会被打断，因此操作系统还要执行**进程管理**，以跟踪进程的进展以及所有中间状态。

内存管理和进程管理都需要**CPU调度**，即确定某个时刻CPU要执行内存中的哪个进程。

内存管理、进程管理和CPU调度是本章的三个讨论重点。其他关于操作系统的重要主题（如文件管理和二级存储）将留待第11章讨论。

记住，操作系统自身也是必须执行的程序，所以在内存中也要和其他系统软件及应用程序一起管理和维护OS进程。执行OS的CPU就是执行其他程序的CPU，因此也要把OS进程排入竞争CPU的队列中。

在深入探讨资源（如主存和CPU）管理前，还需要介绍一些一般的概念。

多道程序设计（multiprogramming）：同时在主存中驻留多个程序，由它们竞争CPU的技术。
内存管理（memory management）：了解主存中载有多少个程序以及它们的位置的动作。
进程（process）：程序执行过程中的动态表示法。
进程管理（process management）：了解活动进程的信息的动作。
CPU调度（CPU scheduling）：确定主存中的哪个进程可以访问CPU以便执行的动作。

10.1.2 批处理

20世纪60年代和70年代典型的计算机是放置在专用空调房中的大机器。它的处理是由操作员管理的。用户需要把自己的程序交付给操作员才能执行它，通常采用的是一叠穿孔卡片。然后用户再回来取打印出的结果，不过可能是第二天才能取了。

在交付程序时，用户需要为执行程序所需的系统软件或其他资源提供一套单独的指令。程序和系统指令集合在一起，称为作业。操作员要启动所有必需的设备，按照作业中的要求载入特定的系统软件。因此，在这些早期计算机上，为执行程序做准备是个耗时的过程。

为了更有效地执行这一过程，操作员会把来自多个用户的作业组织成分批。一个分批包含一组需要相同或相似资源的作业，操作员从而不必反复地载入和准备相同的资源。图10-2展示了这一过程。

可以在多道程序设计的环境中执行分批系统。在这种情况下，操作员将把一个分批中的多个作业载入内存，这些作业将竞争CPU和其他共享资源的使用权。当作业具备了所需的资源后，将被调度使用CPU。

虽然批处理的原始概念并不属于现代操作系统的功能，但是这一概念被保留了下来。现在术语“批”表示的是一个系统，在这个系统中，程序和系统资源的协作与执行不需用户和

程序之间的交互。现代操作系统中的批处理概念，是允许用户把一系列OS命令定义为一个批文件，以控制一个大型程序或一组交互程序的处理。例如，MS Windows中具有.bat后缀的文件就源自于批控制文件的想法，它们存放的是系统命令。

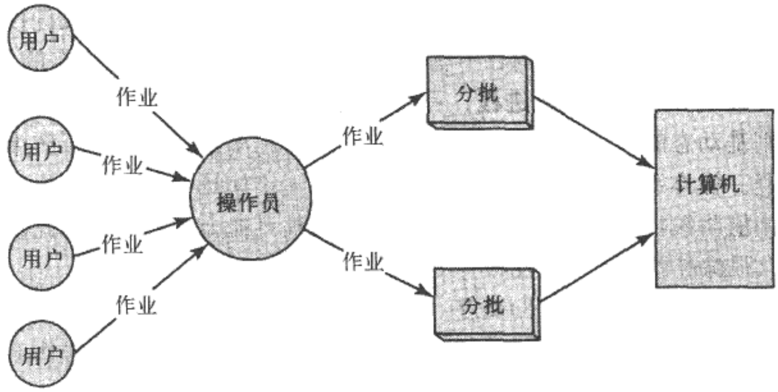


图10-2 在早期的系统中，操作员需要分批组织作业

尽管目前使用的大多数计算机都是交互式的，但有些作业仍然会自行批处理。例如，一个公司的月薪处理就是这样一项使用特定资源的大型作业，它并不需要人机交互。

早期的批处理允许多个用户共享一台计算机。虽然随着时间的迁移，批处理的重点已经改变了，但是批处理系统却给我们就资源管理留下了宝贵的经验。早期计算机系统的操作员扮演的角色，正是现代操作系统软件所做的。

10.1.3 分时操作

第1章提到过，如何更大程度地利用机器的能力和速度的问题引出了分时的概念。**分时系统**允许多个用户同时与计算机进行交互。多道程序设计法允许同时有多个活动进程，从而给了程序员直接与计算机系统交互，且仍然共享资源的能力。

分时系统创建了每个用户都专有这台计算机的假象。也就是说，每个用户都不必主动竞争资源，尽管幕后的事实还是如此。用户可能知道他在和其他用户共享这台机器，但不必为此付出额外的操作。操作系统负责在幕后管理资源（包括CPU）共享。

单词“虚拟”的意思是有效但并不存在的。在分时系统中，每个用户都有自己的**虚拟机**，可以使用虚拟机中的所有资源（都是有效的）。但其实这些资源是由多个用户共享的。

分时系统最初由一台**主机**和一组连接到主机的哑终端构成。**哑终端**只是一个显示器和一个键盘。用户坐在终端前，登录到主机。哑终端可以遍布整幢大楼，而主机则放置在专用的房间中。操作系统驻留在主机中，所有处理都在这里发生。

> **分时系统**（timesharing）：多个交互用户同时共享CPU时间的系统。
> **虚拟机**（virtual machine）：分时系统创建的每个用户都有专有机器的假象。
> **主机**（mainframe）：一个大型的多用户计算机，通常与早期的分时系统相关。
> **哑终端**（dumb terminal）：在早期的分时系统中用户用于访问主机的一套显示器和键盘。

每个用户由主机上运行的一个*登录进程*表示。当用户运行程序时，将创建另一个进程（由用户的登录进程生成）。CPU时间由所有用户创建的所有进程共享。每个进程将顺次得到

一小段CPU时间。前提是CPU足够快，能够处理多个用户的请求并不使任何用户发现自己在等待。事实上，分时系统的用户有时会发现系统响应减慢了，这是由活动用户的数量和CPU的能力决定的。也就是说，当系统负荷过重时，每个用户的机器看来都变慢了。

虽然主机是过时的概念，但分时概念却不是。目前，许多台式计算机运行的操作系统都以分时的方式支持多个用户。尽管事实上只有一个用户坐在计算机前，但其他用户可以用其他计算机通过网络连接到这台计算机上。

10.1.4 其他OS要素

随着计算技术的不断提高，机器自身体积也变得越来越小。大型计算机演变成了*小型机*，这种机器不再需要专用的放置空间。小型机成为了分时系统的基础硬件平台。*微型机*则第一次采用单个的集成芯片作为CPU，成了真正可以放在书桌上的计算机，从而引发了*个人计算机*（PC）的想法。顾名思义，个人计算机不是为多个用户设计的，最初的个人计算机操作系统反映出了这种简单性。随着时间的推移，个人计算机无论在功能还是在与大型系统（如分时）协作方面都有了长足的发展。虽然常常用PC称呼台式计算机，但有时也使用术语*工作站*，这种叫法可能更恰当，说明它一般是服务于个人的，不过也能够支持多个用户。操作系统已经发展成支持计算机用法的这些变化了。

操作系统还必须把计算机通常要连接到网络这个因素考虑在内。目前，我们通过万维网进行网络通信。虽然后面的章节才会讨论网络通信，但是这里必须承认网络通信带给操作系统的影响，这样的通信方式仍然是OS必须支持的。

操作系统要负责与各种各样的设备通信。通常，这些通信是在设备驱动程序的协助下完成的，所谓*设备驱动程序*，即了解特定设备接收和发布信息所希望采用的方式的小程序。通过使用设备驱动程序，操作系统就不必对所有可能与之通信的设备都了如指掌。这是另一个成功的抽象实例。新硬件通常会附带适用的驱动程序，从制造商的网站上一般可以下载到最新的驱动程序。

操作系统的最后一个要素是需要支持*实时系统*的。所谓**实时系统**，就是必须给用户提供最少**响应时间**的系统。也就是说，必须严格控制收到信号和生成响应之间的延迟。实时响应对某些软件至关重要，如机器人控制、核反应堆控制或导弹控制等。尽管所有操作系统都知道响应时间的重要性，但是实时操作系统则更加致力于优化这个方面。

> **实时系统**（real-time system）：应用程序的特性决定了响应时间至关重要的系统。
> **响应时间**（response time）：收到信号和生成响应之间的延迟时间。

10.2 内存管理

让我们来回顾一下第5章中对主存的介绍。所有程序在执行时都存储在主存中。这些程序引用的数据也都存储在主存中，以便程序能够访问它们。可以把主存看作一个大块的连续空间，这个空间被分成了8位、16位或32位的小组。主存中的每个字节或字有一个对应的地址，这个地址只是一个整数，唯一标识了内存中的一个特定部分。如图10-3所示。第一个主存单元的地址是0。

本章前面介绍过多道程序设计环境，也就是在主存中同时驻留多个程序（和它们的数据）。因此，操作系统必须采用下列技术：

• 跟踪一个程序驻留在内存的什么位置以及是如何驻留的。

• 把逻辑程序地址转换成实际的内存地址。

程序中到处都是对变量的引用和对程序其他部分的引用。在编译程序时，这些引用将被转换成数据或代码驻留的内存地址。但是我们并不确切地知道程序载入了主存中的什么位置，那么如何知道使用什么地址呢？

解决方法是使用两种地址——逻辑地址和物理地址。**逻辑地址**（有时又叫做虚拟地址或相对地址）是指定了一个普通地址的值，这个地址是相对于程序，而不是相对于主存的。**物理地址**是主存储设备中的真实地址，如图10-3所示。

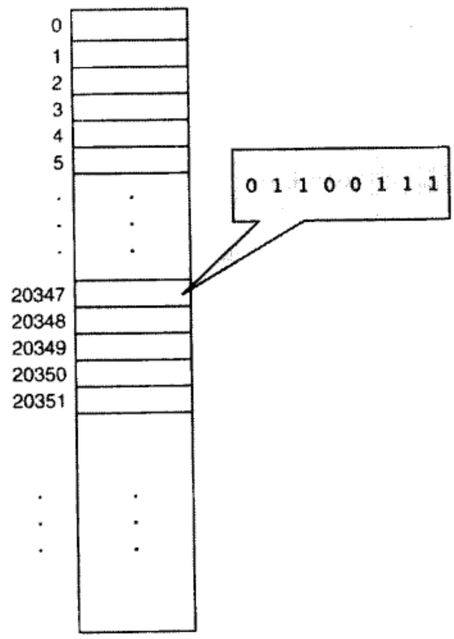


图10-3　主存是由特定地址引用的连续的位集合

逻辑地址（logical address）：对一个存储值的引用，是相对于引用它的程序的。

物理地址（physical address）：主存储设备中的真实地址。

在编译程序时，对标识符（如变量名）的引用将被转化为逻辑地址。当程序最终载入内存时，每个逻辑地址将被转换成对应的物理地址。逻辑地址和物理地址间的映射叫做**地址联编**。把逻辑地址联编到物理地址的时间越迟，得到的灵活度越大。逻辑地址使得程序可以在内存中移动，或者每次载入不同的位置。只要知道程序存储的位置，就可以确定任何逻辑地址对应的物理地址。为了简化本章的实例，我们采用十进制进行地址联编计算。

地址联编（address binding）：逻辑地址和物理地址间的映射。

下面的几节将分析以下三种技术的基本原理：

• 单块内存管理

• 分区内存管理

• 页式内存管理

10.2.1　单块内存管理

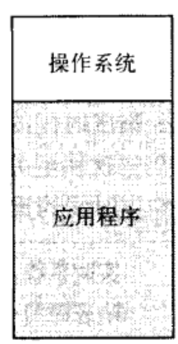


图10-4　分成了两部分的主存

假设内存中只有两个程序——操作系统和要执行的应用程序，这样可以使问题简单一些。我们把主存分为两部分，每个程序占用一部分，如图10-4所示。操作系统得到了所需要的空间，余下的分配给了应用程序。

这种方法称为**单块内存管理**法，因为整个应用程序被载入了一大块内存中。除了操作系统外，一次只能处理一个程序。进行地址联编所要做的只是把操作系统的地址考虑在内。

单块内存管理（single contiguous memory management）：把应用程序载入一段连续的内存区域的内存管理方法。

在这种内存管理机制中，逻辑地址只是一个相对于程序起始位置的整数值。也就是说，就像程序载入主存的地址是0一样。因此，要生成物理地址，只要用逻辑地址加上程序在物理主存中的起始地址即可。

让我们解释得更清楚一点。如果载入程序的起始地址是A，那么逻辑地址L对应的物理地址就是A+L。如图10-5所示。用实数来解释会更加清楚。假设载入程序的内存起始地址是555555。如果程序使用的相对地址是222222，那么它引用的物理内存地址就是777777。

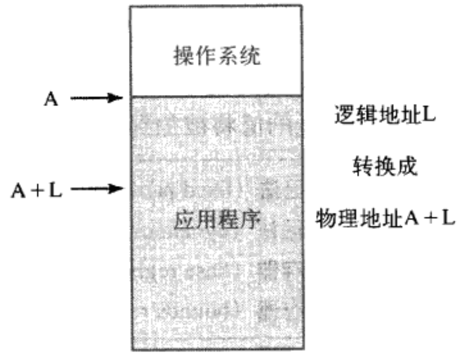


图10-5 逻辑地址和物理地址的联编

至于地址L是什么无关紧要。只要知道程序的起始地址A，就可以把逻辑地址转换成物理地址。

你也许会说，如果交换操作系统和应用程序的位置，那么应用程序的逻辑地址就应该等于物理地址了。不错，但是这样就会有其他问题。例如，内存管理机制必须考虑安全问题。尤其是在多道程序设计环境中，必须防止一个程序访问未分配给它的内存空间。把操作系统载入地址0处，那么应用程序就可以使用所有的逻辑地址，除非它们超过了主存自身的限制。如果把操作系统移到程序之后，就必须确保逻辑地址不会访问操作系统的内存空间。尽管这种操作并不难，但却增加了处理的复杂度。

单块内存管理法的优点在于实现和管理都很简单，但却大大浪费了内存空间和CPU时间。应用程序一般不可能需要操作系统剩余的所有空间，而且在程序等待某些资源的时候，还会浪费CPU时间。

10.2.2 分区内存管理

稍微复杂一些的内存管理方法是同时在内存中驻留多个应用程序，共享内存空间和CPU时间。因此，内存不止再分成两部分。有两种划分内存的方法——固定分区法和动态分区法。使用**固定分区法**，主存将被划分为特定数目的分区。这些分区的大小不一定要相同，但在操作系统初始引导时它们的大小就固定了。作业将被载入空间足够容纳它的分区。OS具有一个地址表，存放了每个分区的起始地址和长度。

使用**动态分区法**，将根据程序的需要创建分区。初始时，主存将被看作一个大的空白分区。当载入程序时，将从主存划分出一块刚好能容纳程序的空间，留下一块新的、小一些的空白分区，以便之后供其他程序使用。操作系统将维护一个分区信息表，不过在动态分区中，地址信息会随着程序的载入和清除而改变。

无论是固定分区还是动态分区，任何时候内存都是被划分为一组分区，有些是空的，有些分配给了程序。如图10-6所示。

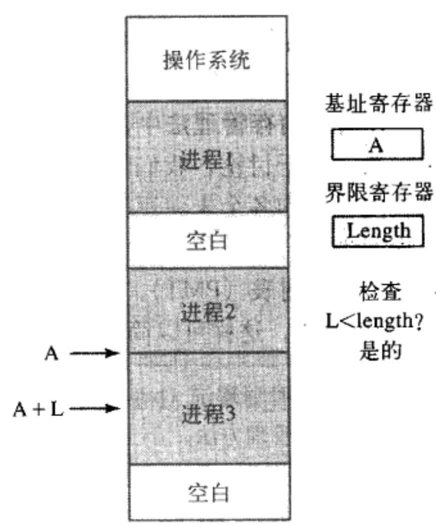


图10-6 分区内存管理法中的地址解析

固定分区和动态分区的地址联编基本上是一样的。与单块内存管理法一样，逻辑地址是相对于0起始点的整数。OS处理地址转换的方式有很多。一种方法是使用CPU中的两个专用寄存器帮助管理寻址。当CPU开始运行一个程序时，OS将把它的分区起始地址存储到**基址寄存器**中。同样地，分区的长度将被存入**界限寄存器**。当逻辑地址被引用时，首先它将与界限寄存器中的值进行比较，确保该引用属于分配给程序的内存空间。如果引用没有超出范围，那么逻辑地址的值将被加到基址寄存器中，以生成物理地址。

> **固定分区法**（fixed-partition technique）：把内存分成特定数目的分区以载入程序的内存管理方法。
> **动态分区法**（dynamic-partition technique）：根据容纳程序的需要对内存分区的内存管理方法。
> **基址寄存器**（base register）：存放当前分区的起始地址的寄存器。
> **界限寄存器**（bounds register）：存放当前分区的长度的寄存器。

那么，对于一个新程序，应该分配给它哪个分区呢？下面有三种常用的分区选择法：

- *最先匹配*，即把第一个足够容纳程序的分区分配给它。
- *最佳匹配*，即把最小的能够容纳程序的分区分配给它。
- *最差匹配*，即把最大的能够容纳程序的分区分配给它。

在固定分区法中，最差匹配没有意义，因为它将浪费较大的分区。最先匹配和最佳匹配适用于固定分区。但在动态分区中，最差匹配常常是最有用的，因为它留下了最大可能的空白分区，可以容纳之后的其他程序。

当程序终止时，分区表将被更新，以反映这个分区现在是空白的，新程序可以使用它了。在动态分区中，连续的空白分区将被合并成一个大的空白分区。

分区内存管理法同时把几个程序载入内存，可以有效地利用主存。但要记住，一个分区必须要能够容纳整个程序。虽然固定分区比动态分区容易管理，但却限制了进来的程序的机会。系统本身可能有足够的空间容纳这些程序。在动态分区中，作业可以在内存中移动，以创建较大的空白分区。这个过程叫做压缩。

10.2.3　页式内存管理

页式内存管理法需要跟踪分配的内存，还要解析地址，从而给操作系统增加了很多负担。但是，这种方法提供的好处值得做出这些牺牲。

在**页式内存管理法**中，主存将被分成小的大小固定的存储块，叫做**帧**。进程将被划分为**页**，为了便于讨论，我们假设页的大小等于帧的大小。在程序执行时，进程的页将被载入分散在内存中的各个未使用的帧中，因此，一个进程的页可能是四处散落的、无序的，与其他进程的页混合在一起。为了掌握进程页的分布，操作系统需要为内存中的每个进程维护一个独立的**页映射表**（PMT），把每个页映射到载入它的帧。如图10-7所示。注意，页和帧都是从0开始编号的，这样可以简化地址的计算。

> **页式内存管理法**（paged memory technique）：把进程划分为大小固定的页，载入内存时存储在帧中的内存管理方法。
> **帧**（frame）：大小固定的一部分主存，用于存放进程页。
> **页**（page）：大小固定的一部分进程，存储在内存帧中。
> **页映射表**（page map table, PMT）：操作系统用于记录页和帧之间的关系的表。

页式内存管理系统中的逻辑地址与分区系统中的一样，都是一个相对于程序起始点的整数值。但这个地址将被转换成两个值——页编号和偏移量。用页面大小除逻辑地址得到的商是页编号，余数是偏移量。因此，如果页面大小是1024，那么逻辑地址2566对应的就是进程的第2页的第518个字节。逻辑地址通常被表示为<页编号，偏移量>，如<2,518>。

要生成物理地址，首先需要查看PMT，找到页所在的帧的编号，然后用帧编号乘以帧大小，加上偏移量即可。例如图10-7中的例子，如果进程1是活动的，逻辑地址<1,222>将被进行如下处理：进程1的页面1存储在帧12中，因此这个逻辑地址对应的物理地址是$12 \times 1024 + 222 = 12\ 510$。注意，有两种逻辑地址是无效的，一种是越过了进程的界限，一种是偏移量大于帧大小。

分页的优点在于不必再把进程存储在连续的内存空间中。这种分割进程的能力把为进程寻找一大块可用空间的问题转化成了寻找足够多的小块内存。

页式内存管理思想的一个重要扩展是**请求分页**思想，它利用了程序的所有部分不必同时处于内存中这一事实。任何时刻CPU都只访问进程的一个页面，此时，进程的其他页面是否在内存中无关紧要。

P1 PMT

帧	页
0	5
1	12
2	15
3	7
4	22

P2 PMT

帧	页
0	10
1	18
2	1
3	11

内存

帧	内容
0	
1	P2/Page2
2	
3	
4	
5	P1/Page0
6	
7	P1/Page3
8	
9	
10	P2/Page0
11	P2/Page3
12	P1/Page1
13	
14	
15	P1/Page2

⋮

图10-7 页式内存管理法

在请求分页法中，页面经过要求才会被载入内存。也就是说，当引用一个页面时，首先要看它是否已经在内存中了，如果该页面在内存中，就完成访问，否则，要从二级存储设备把这个页面载入可用的帧，然后再完成访问。从二级存储设备载入页面通常会引发把其他页面写回二级存储设备，这种行为叫做**页面交换**。

> **请求分页**（demand paging）：页式内存管理法的扩展，只有当页面被引用（请求）时才会被载入内存。
>
> **页面交换**（page swap）：把一个页面从二级存储设备载入内存通常会使另一个页面从内存中删除。

请求分页法带来了**虚拟内存**的思想，即对程序大小没有任何限制的假象（因为整个程序不必同时处于内存中）。在前面分析的所有内存管理法中，整个进程都必须作为整体载入内存。因此，进程大小始终有一个上限。请求分页法消除了这一限制。

不过，虚拟内存在程序执行时需要很多开销。以前，一旦程序载入了内存，就完全处于内存中，准备好执行了。采用虚拟内存法，则经常需要在主存和二级存储设备间进行页面交换。当一个进程等待页面交换时，另一个进程接管CPU的控制，那么这种开销是可以接受的。但页面交换过多叫做**系统颠簸**，会严重降低系统的性能。

> **虚拟内存**（virtual memory）：由于整个程序不必同时处于内存而造成的程序大小没有限制的假象。
>
> **系统颠簸**（thrashing）：频繁的页面交换造成的低效处理。

10.3 进程管理

操作系统必须管理的另一个重要资源是每个进程使用的CPU时间。要理解操作系统是如何管理进程的，必须了解进程在生存周期中的各个阶段，理解使进程在计算机系统中正确运行所要管理的信息。

10.3.1 进程状态

在计算机系统的管理下，进程会历经几种状态，即进入系统、准备执行、执行、等待资源以及执行结束。图10-8展示了**进程状态**。每个方框表示一种进程状态，方框之间的箭头说明了一个进程如何以及为什么从一种状态转移到另一种状态。

> **进程状态**（process state）：在操作系统的管理下，进程历经的概念性阶段。

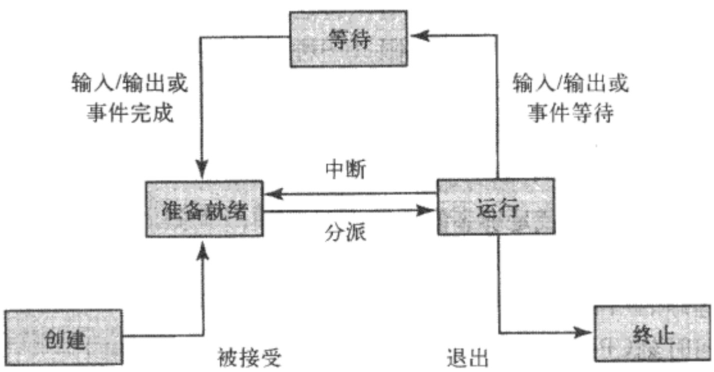


图10-8　进程的生命周期

下面来分析在进程的每个阶段会发生哪些事情。

在*创建阶段*，将创建一个新进程。例如，可能是由用户登录到一个分时系统创建了一个登录进程，也可能是在用户提交程序后创建了一个应用进程，或者是操作系统为了完成某个特定的系统任务而创建了一个系统进程。

在*准备就绪状态*中，进程没有任何执行障碍。也就是说，准备就绪状态下的进程并不是在等待某个事件发生，也不是在等待从二级设备载入数据，而只是等待使用CPU的机会。

*运行状态*下的进程是当前CPU执行的进程。它的指令将按照读取-执行周期被处理。

*等待状态*下的进程是当前在等待资源（除了CPU以外的资源）的进程。例如，一个处于等待状态的进程可能在等待从二级设备载入一个页面，也可能在等待另一个进程给它发送信号，以便继续执行。

*终止状态*下的进程已经完成了它的执行，不再是活动进程。此时，操作系统不再需要维护有关这个进程的信息。

注意，可能同时有多个进程处于准备就绪或等待状态，但只有一个进程处于运行状态。

在创建进程后，操作系统将接纳它进入准备就绪状态。在得到CPU调度算法的指示后，进程将被分派到运行状态。（在10.4节中将详细讨论CPU调度算法。）

在运行过程中，进程可能被操作系统中断，以便另一个进程能够获得CPU资源。在这种情况下，进程将返回准备就绪状态。正在运行的进程还可以请求一个未准备好的资源，或者

请求I/O读取新引用的部分进程，在这种情况下，它将被转移到等待状态。正在运行的进程最后将得到足够的CPU时间以完成它的处理，正常终止；或者将生成一个无法解决的错误，异常终止。

当等待中的进程得到了它在等待的资源后，它将再次转移到准备就绪状态。

10.3.2 进程控制块

操作系统必须为每个活动进程管理大量的数据。这些数据通常存储在称为**进程控制块**（PCB）的数据结构中。通常，每个状态由一个PCB列表表示，处于该状态的每个进程对应一个PCB。当进程从一个状态转移到另一个状态时，它对应的PCB也会从一个状态的列表中转移到另一个状态的列表。新的PCB是在最初创建进程（新状态）的时候创建的，将一直保持到进程终止。

> **进程控制块**（process control block）：操作系统管理进程信息使用的数据结构。

PCB存储了有关进程的各种信息，包括程序计数器的当前值（说明了进程中下一条要执行的指令）。如进程的生命周期所示，进程在执行过程中可能会被中断多次。每次中断时，它的程序计数器的值将被保存起来，以便当它再次进入运行状态时可以从中断处开始执行。

PCB还存储了进程在其他所有CPU寄存器中的值。记住，只有一个CPU，因此只有一套CPU寄存器。这些寄存器存放的是当前执行的进程的值（运行状态只有一个进程）。每当一个进程进入了运行状态，当前正在运行的进程的寄存器值将被存入它的PCB，新运行的进程的寄存器值将被载入CPU。这种信息交换叫做**上下文切换**。

> **上下文切换**（context switch）：当一个进程移出CPU，另一个进程取代它时发生的寄存器信息交换。

PCB还要维护关于CPU调度的信息，如操作系统给予进程的优先级。它还包括内存管理的信息，如（分区系统的）基址寄存器和界限寄存器的值或（页式系统的）页表。最后，PCB还具有核算信息，如账户、时间限制以及迄今为止使用的CPU时间。

10.4 CPU调度

所谓CPU调度，就是确定把哪个处于准备就绪状态的进程移入运行状态。也就是说，CPU调度算法将决定把CPU给予哪个进程，以便它能够运行。

CPU调度可以是在一个进程从运行状态转移到等待状态或终止时发生的。这种类型的CPU调度叫做**非抢先调度**，因为对新的CPU进程的需要是当前执行进程的活动的结果。

CPU调度还可以是在一个进程从运行状态转移到准备就绪状态或一个进程从等待状态转移到准备就绪状态时发生的。它们属于**抢先调度**，因为当前运行的进程被操作系统抢占了。

> **非抢先调度**（nonpreemptive scheduling）：当当前执行的进程自愿放弃了CPU时发生的CPU调度。
>
> **抢先调度**（preemptive scheduling）：当操作系统决定照顾另一个进程，抢占当前执行进程的CPU资源时发生的CPU调度。

通常用特殊的标准（如**周转周期**）来评估调度算法。所谓周转周期，是从进程进入准备就绪状态到它退出运行状态的时间间隔。进程的平均周转周期越短越好。

> **周转周期**（turnaround time）：从进程进入准备就绪状态到它完成之间的时间间隔，是评估CPU调度算法的标准之一。

用于确定从准备就绪状态首选哪个进程进入运行状态的方法有很多。在下面的小节中将分析其中三种方法。

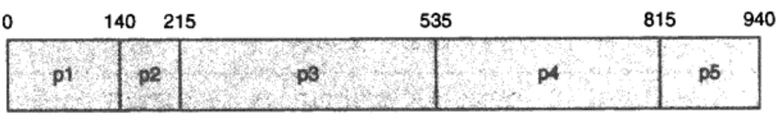
10.4.1 先到先服务

在先到先服务（FCFS）调度方法中，进程按照它们到达准备就绪状态的顺序转移到CPU。FCFS调度是非抢先的。一旦进程获得了CPU的访问权，那么除非它自动请求转入等待状态（如请求其他进程正在使用的设备），否则将一直占用CPU。

假设进程p1到p5几乎同时到达准备就绪状态（为了简化计算），但它们仍然有进入顺序，具有特定的服务时间，如右表所示。

进　程	服务时间
p1	140
p2	75
p3	320
p4	280
p5	125

在FCFS调度方法中，每个进程将依次访问CPU。为了简单起见，我们假设这些进程不会自行请求等待。下面的Gantt图说明了完成进程的顺序和时间。



由于我们假设所有进程同时到达，所以每个进程的周转周期等于它的完成时间。这里的平均周转周期是 $(140+215+535+815+940)/5=529$。

事实上，进程并非同时到达的。在这种情况下，平均周转周期的计算方法是一样的，只是需要考虑每个进程的到达时间。每个进程的周转周期是它的完成时间减去到达时间。

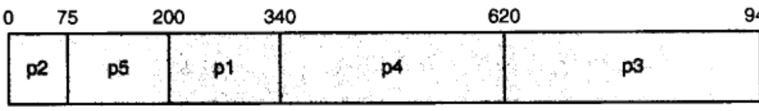
FCFS算法很容易实现，但却因不注意某些重要因素（如服务时间的需求）而变得复杂。虽然我们在计算周转周期的时候使用了服务时间，但是FCFS算法却没有用这些信息来帮助确定最佳的进程调度顺序。

> **训练记录**
>
> 耐克和苹果公司宣布了一项合作，即让运动鞋能够和iPod进行无线对话。耐克公司把传感器和一个无线装置放入选中的运动鞋中。有了这个系统，跑步者就能够记录每次训练的距离、时间、速度和消耗的卡路里。当跑步者完成了训练时，这些数据就会由苹果公司的iTune音乐软件下载下来，存储到耐克公司维护的网站上，以供分析。iTune音乐商店也提供了一个运动专栏，其中具有特别的音乐和著名运动员的训练记录。

10.4.2 最短作业优先

最短作业优先（SJN）CPU调度算法将查看所有处于准备就绪状态的进程，分派一个具有最短服务时间的。和FCFS一样，它通常被实现为非抢先算法。

下面是FCFS示例中使用过的一套进程的Gantt图。由于选择标准不同，调度和完成进程的顺序也就不同。



这个示例的平均周转周期是 (75 + 200 + 340 + 620 + 940)/5 = 435。

注意，SJN算法是基于未来信息的。也就是说，它将把CPU给予执行时需要最短时间的作业。这个时间基本上是不可能确定的。因此运行这个算法，每个进程的服务时间是操作系统根据各种概率因素和作业类型推算的。但如果估计错误，算法的前提就崩溃了，它的性能将恶化。SJN算法是*可证明最佳的*，意思是如果知道每个作业的服务时间，那么相对于其他算法来说，SJN算法能使所有作业生成最短的周转周期。但是，由于我们不可能绝对地明了未来，所以我们只能猜测并且希望这种猜测是正确的。

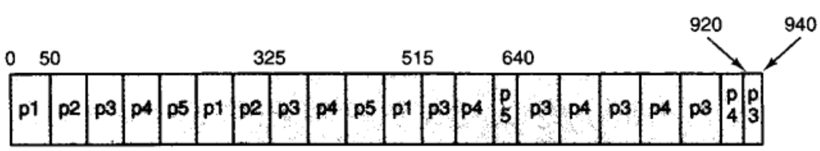
10.4.3 循环调度法

CPU的循环调度法将把处理时间平均分配给所有准备就绪的进程。该算法将建立单独的**时间片**（或时间量子），即在每个进程被抢占并返回准备就绪状态之前收到的时间量。被抢占的进程最终会得到其他的CPU时间片。这个过程将持续到进程得到了所需的全部时间从而终止了为止。

> **时间片**（time slice）：在CPU循环调度算法中分配给每个进程的时间量。

注意，循环调度算法是抢先的。时间片到期，进程就会被强制移出CPU，即从运行状态转移到准备就绪状态。

假设一个循环调度算法使用的时间片是50，仍然使用前面示例中的进程集合。Gantt图如下所示：



每个进程都将得到长度为50的时间片，除非它不需要一个完整的时间片。例如，进程2最初需要75个时间单位，开始它将得到50个时间单位。当轮到它再次使用CPU时，它只需要25个时间单位，因此，进程2将终止，在325个时间单位处放弃了CPU。

这个例子的平均周转周期是 (515 + 325 + 940 + 920 + 640)/5 = 668。注意，这个例子的周转周期比其他例子的长。这意味着循环调度法没有其他调度算法好吗？不是的，我们不能只根据一个例子就得出这样的一般结论。只能说，对于某套特定的进程，一种算法比另一种算法有效。算法有效性的一般分析要复杂得多。

CPU的循环调度算法可能是应用最广泛的。它一般支持所有的作业，被认为是最公平的算法。

Steve Jobs



Steve Jobs生于1955年，使他闻名于世的可能就是1976年他与Steve Wozniak和Ronald Wayne共同创建了苹果公司。当时，大多数计算机要不就是大型机（有些与一个小房间一样大），要不就是微电脑（大概与冰箱一样大），无任何用户友好性可言，几乎只能用于大型商业公司。Jobs预见到了个人计算机的流行趋势。他常常因使计算机平民化而受人赞誉。

Jobs和Wozniak在Jobs的卧室中设计了Apple I，并在他父母的车库中造出了它。Jobs和Wozniak分别卖掉了他们的奖品（一辆Volkswagen面包车和一个惠普的科学计算器），得到了1300美元的资本，从而建立了他们的公司。四年后，苹果公司上市了。第一个交易日结束时，它的市值高达12亿美元。

Jobs领导了开发Apple Macintosh（以McIntosh apple命名）的小组，这可能也是苹果公司最著名的机器了。Macintosh是第一个在商业上获得成功的计算机，它有图形化的用户界面和鼠标。在Macintosh发布不久，Jobs就由于和苹果公司当时的CEO John Sculley的权力之争，被挤出了苹果公司。

在被迫离开自己创建的公司后，Jobs创建了另一家计算机公司NeXT，这家公司于1996年被苹果公司以4亿200万美元的价格收购了。这一收购案不仅使Jobs回到了他自己的公司，还使他成为了苹果公司的CEO。在他的领导下，苹果公司又发布了iMac，这款计算机被誉为"桌面计算系统的黄金标准"。

1986年，Jobs收购了一家计算机图形公司，将其命名为Pixar，从而进入了计算机动画制作领域。Pixar公司制作了多个票房热点电影，如《虫虫特工队》（A Bug's life）、《玩具总动员》（Toy Story）、《怪物公司》（Monsters Inc.）、《海底总动员》（Finding Nemo）。

Jobs自己没有完成大学学业，2005年在斯坦福大学的毕业典礼上，他在演讲中给了毕业生下面的忠告："你即将开始追寻你所热爱的事业"。

小结

操作系统是管理计算机资源的系统软件的一部分，是人类用户、应用软件和系统硬件之间的协调者。

多道程序设计技术允许在内存中同时驻留多个程序，让它们竞争CPU时间。进程是执行中的程序。操作系统必须执行精细的CPU调度、内存管理和进程管理，以确保访问的公平性。

批处理将把使用相同或相似资源的作业组织成批。分时技术将为每个用户创建一个虚拟机，允许多个用户同时与计算机进行交换。

操作系统必须管理内存，以控制和监管把进程载入主存中的什么位置。任何内存管理技术都必须定义联编逻辑地址和物理地址的方法。有多种内存管理的策略。单块内存管理法除了操作系统外只允许一个程序驻留主存。分区法是把主存划分成几个分区，进程要载入这些分区。固定分区法中的分区个数是固定的，动态分区法则是根据载入的进程的需要决定的。页式内存管理法是把内存划分为帧，把程序划分为页。程序的页在内存中不必是连续的。请求分页法在任何时刻都只需要一部分程序位于内存中。

操作系统还要管理进程的生命状态，即程序在执行过程中要历经的阶段。进程控制块存储了每个进程的必要信息。

CPU调度算法确定了下一个使用CPU的进程。先到先服务的CPU调度法给予最早达到的作业优先权。最短作业优先算法给予运行时间最短的作业优先权。循环调度算法让每个活动进程轮流使用CPU，每个进程得到一个小时间片。

道德问题：数字版权管理和关于Sony公司的根目录案件的争论

什么是数字版权管理（Digital Right Management, DRM）技术？所谓DRM，指的是内容所有者用于“控制数据（如软件、音乐和电影）和硬件访问”的一组技术（摘自Wikipedia，2006）。有了DRM技术，内容提供者和软件开发商就能够在数字媒体中嵌入代码，以控制自己产品的使用权。

DRM系统的拥护者认为这项技术可以防止某些用户侵犯版权。但是许多DRM技术的批评者则置疑使用DRM技术强化版权法的方式。例如，某些法律专业人士就认为DRM系统可能冒犯了版权法的公平使用的规定。另一些批评家则担心与传统的版权保护机制提供的控制相比，DRM技术能让内容所有者对数字媒体的版权进行过度的控制。出于这些原因，自由软件基金会（Free Software Foundation，FSF）的创始人Richard Stallman认为，把DRM看作一种“数据约束管理”技术更好。

还有一些批评家担心内容所有者会滥用DRM系统来控制用户的计算机（在后台），甚至会被公司用来监视受信任的用户。近来，Sony BMG音乐娱乐公司使用DRM系统调用扩大版权保护（eXtended Copy Protection，XCP）软件来保护自己的音乐CD，在出现这个案例后，这种担心变得越来越明显了。

Sony公司于2005年10月发生的一起事件引起了广泛的关注，当时一个博客主写了一篇文章，列出了Sony公司的版权保护软件设计上的一些缺陷，这些缺陷是一些安全漏洞，恶意的软件程序（包括病毒和蠕虫）可以利用这些安全漏洞。这个博客主还指出，Sony公司并未提供卸载程序来删除这个XCP软件。这个缺陷公布于众不久，Sony公司就发布了一个工具，使得用户可以删除这个具有争议的软件。遗憾的是，Sony公司的这个删除工具暴露了XCP组件的根目录中隐藏的文件（它并没有删除根目录自身）。这一暴露引起了更多关于隐私权和安全性的关注。Sony公司最终给这个删除工具发布了一个更新版本，使得用户可以成功地卸载根目录。

有些批评家认为Sony公司通过这个XCP系统已经冒犯了客户的隐私权，它使用代码在客户的机器上创建了一个“后门”。另一些批评家认为，Sony公司的DRM程序确实违反了版权法。为了应对这些批评家，Sony公司决定撤回这种版权保护软件，它召回了店面中所有未售出的CD，还允许客户把已经购买的CD换成没有XCP软件的版本。Sony公司打算弥补这些问题，但是并没有让所有的批评家闭口。Sony BMG公司受到了多个联合诉讼，其中包括来自加利福尼亚、纽约和得克萨斯的诉讼。

Sony公司的根目录案件引发了下列问题。某些DRM系统真的违反了版权法，而不是保护了它吗？它们冒犯了个人隐私权吗？普通用户可以信任像Sony公司这样的内容所有者吗？它们可以使用DRM技术轻松地监视用户以及控制他们计算机的某些方面。Sony公司使用的这类DRM系统能够说明这些公司需要DRM系统来保护它们的知识产权吗？

练习

判断练习1～18中的陈述的对错：

A. 对　　B. 错

1. 操作系统是一种应用软件。
2. 操作系统提供了基本的用户界面，使用户能够

使用计算机。

3. 计算机可以具有多个操作系统，但任何时刻都只有一个操作系统控制机器。
4. 多道程序设计是使用多个CPU运行程序的技术。
5. 在20世纪60年代和70年代期间，操作员会把类似的计算机作业组织成批来运行。
6. 批处理意味着用户和程序间的高级交互。
7. 分时系统允许多个用户同时与一台计算机进行交互。
8. 所谓哑终端，是连接到主机上的I/O设备。
9. 逻辑地址是真正的内存地址。
10. 单块内存管理系统中的地址由页编号和偏移量构成。
11. 在固定分区系统中，主存将被划分为几个大小相同的分区。
12. 界限寄存器存放的是分区的结束地址。
13. 页式内存管理系统中的第一个页面是页面0。
14. 处于运行状态的进程是CPU当前执行的进程。
15. 进程控制块（PCB）是存储一个进程的所有信息的数据结构。
16. CPU调度方法决定了内存中有哪些程序。
17. 先到先服务调度算法是可证明最佳的CPU调度算法。
18. 时间片是循环调度法中每个进程从获得CPU到被抢占之间的时间量。

为练习19～23中的信息找出与之匹配的操作系统。

A. Mac OS　　B. UNIX
C. Linux　　D. DOS
E. Windows

19. Apple计算机采用的是什么操作系统？
20. 在历史上，严谨的程序员一般选用什么操作系统？
21. UNIX的PC版是什么？
22. Microsoft操作系统家族提供的PC版本是什么？
23. 原始的PC操作系统叫什么？

为练习24～26中的定义找出与之匹配的软件类型。

A. 系统软件　　B. 操作系统
C. 应用软件

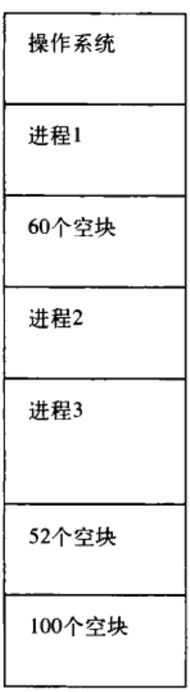
24. 帮助我们解决现实世界问题的程序。
25. 管理计算机系统并与硬件交互的程序。
26. 管理计算机资源并为其他程序提供界面的程序。

练习27～72是问答题或简答题。

27. 请区别应用软件和系统软件。
28. 什么是操作系统？
29. 请解释术语多道程序设计。
30. 下面的术语与操作系统如何管理多道程序设计技术有关。请解释每个术语在这个过程中的角色。
    a) 进程　　b) 进程管理
    c) 内存管理　　d) CPU调度
31. 什么构成了批作业？
32. 从20世纪60年代和70年代的操作员到现在的操作系统，请描述批处理的概念的演变。
33. 定义分时操作。
34. 请说明多道程序设计技术和分时操作之间的关系。
35. 为什么说分时系统中的用户都具有自己的虚拟机？
36. 第7章把虚拟机定义为用于说明真实机器的重要特性的假象机。这一章则把虚拟机定义为分时系统创建的假象，以使每个用户拥有一个专用计算机。请说明这两种定义之间的关系。
37. 分时概念是如何运作的？
38. 什么是实时系统？
39. 什么是响应时间？
40. 请说明实时系统和响应时间之间的关系。
41. 在多道程序设计环境中，可以有多个活动进程。操作系统要管理活动进程的内存需求，必须完成哪些任务？
42. 请区别逻辑地址和物理地址。
43. 什么是地址联编？
44. 列举三种内存管理技术，从中总结出一种通用方法。
45. 何时把一个逻辑地址赋予一个变量？
46. 何时会发生地址联编？
47. 在单块内存管理法中如何划分内存？
48. 在编译程序时，会假设程序载入内存的什么位置？也就是说，假设逻辑地址从何处开始？
49. 在单块内存管理系统中，如果程序被载入地址30215处，（按十进制）计算下列逻辑地址对应的物理地址：
    a) 9223　　b) 2302
    c) 7044

50. 在单块内存管理法中，如果一个变量的逻辑地址是L，程序的起始地址是A，那么联编逻辑地址和物理地址的公式是什么？
51. 在固定分区内存管理系统中，如果基址寄存器的当前值是42993，界限寄存器的当前值是2031，请计算下列逻辑地址对应的物理地址：
    a) 104　　b) 1755
    c) 3041
52. 如果（在固定分区和动态分区中）使用了多个分区，那么基址寄存器存放的是什么？
53. 为什么在计算物理地址前要比较逻辑地址和界限寄存器的值？
54. 在动态分区内存管理系统中，如果基址寄存器的当前值是42993，界限寄存器的当前值是2031，请计算下列逻辑地址对应的物理地址：
    a) 104　　b) 1755
    c) 3041

练习55和56使用的内存状态如下图所示。



55. 如果分区是固定的，到达的新作业需要52个内存块，展示采用下列分区选择法后的内存状态：
    a) 最先匹配　　b) 最佳匹配
    c) 最差匹配
56. 如果分区是动态的，到达的新作业需要52个内存块，展示采用下列分区选择法后的内存状态：
    a) 最先匹配　　b) 最佳匹配
    c) 最差匹配
57. 在页式内存管理系统中，逻辑地址<2,133>的含义是什么？

练习58～60使用的是下列PMT。

页编号	0	1	2	3
帧编号	5	2	7	3

58. 如果帧大小是1024，那么逻辑地址<2,85>对应的物理地址是什么？
59. 如果帧大小是1024，那么逻辑地址<3,555>对应的物理地址是什么？
60. 如果帧大小是1024，那么逻辑地址<3,1555>对应的物理地址是什么？
61. 什么是虚拟内存？它如何应用请求分页？
62. 在操作系统管理下，进程要历经哪些概念性阶段？
63. 请描述进程是如何在各个状态间转换的，给出进程从一种状态转移到另一种状态的明确原因。
64. 什么是进程控制块？
65. OS如何表示每种概念性阶段？
66. 什么是上下文切换？
67. 请区别抢先调度和非抢先调度。
68. 列举并说明三种CPU调度算法。

练习69～72需要使用下表中的进程和服务时间。

进程	P1	P2	P3	P4	P5
服务时间	120	60	180	50	300

69. 采用先到先服务的CPU调度算法，绘制展示每个进程的完成时间的Gantt图。
70. 采用最短作业优先的CPU调度算法，绘制展示每个进程的完成时间的Gantt图。
71. 采用循环调度的CPU调度算法（时间片为60），绘制展示每个进程的完成时间的Gantt图。
72. 请区别固定分区和动态分区。

思考题

1. 第5章说过，控制器就像一个舞台监督，负责组织和管理冯·诺伊曼机的其他部分。操作系统也像一个舞台监督，只是管理范围更大。这个比喻成立吗？
2. OS呈现给用户的界面就像一个具有多扇门的走廊，打开这些门就可以进入住有各种应用程序的房间。从一个房间到另一个房间，必须先返回走廊。采用这种比喻法，可以把文件比喻成什么？时间片又可以比喻成什么？
3. DRM系统真的像DRM技术的支持者们所建议的那样能够保护数字媒体的版权吗？或者像DRM的批评者所说的那样，这些系统有时会违反版权法呢？
4. 某些DRM保护机制允许内容所有者“监视”用户，那么这些DRM系统冒犯了用户的隐私权吗？
5. 从道德上讲，像Sony BMG的XCP版权保护机制这样的DRM系统合理吗？它们应该合法吗？

第11章 文件系统和目录

上一章分析了操作系统扮演的部分角色，特别介绍了进程管理、CPU管理和主存管理。操作系统要管理的另一个关键资源是二级存储设备，通常是磁盘。在日常的计算中，磁盘上文件和目录的组织扮演着关键的角色。文件系统就像摆在桌上的卡片目录，提供了组织良好的信息访问方式。目录结构把文件组织在类和子类中。本章将详细讨论文件系统和目录结构。

目标

学完本章之后，你应该能够：

- 描述文件、文件系统和目录的用途。
- 区别文本文件和二进制文件。
- 根据文件扩展名识别各种文件类型。
- 解释文件类型如何能改进对文件的使用。
- 定义文件的基本操作。
- 比较顺序文件访问法和直接文件访问法
- 讨论与文件保护相关的问题。
- 描述目录树。
- 为目录树创建绝对路径和相对路径。
- 描述几种磁盘调度算法。

11.1 文件系统

第5章说明过主存和二级存储设备间的区别。主存是存放活动的程序和正在使用的数据的地方。主存具有易失性，关掉电源后存储在主存中的信息就会丢失。二级存储设备则具有永久性，即使关闭了电源，它存储的信息依然存在。因此，我们用二级存储设备来永久存储信息。

最常用的二级存储设备是磁盘驱动器，包括计算机主机箱中的硬盘驱动器和能够在计算机间转移使用的便携式软盘。这两种磁盘的基本原理是相同的。其他二级存储设备（如磁带机）主要用于归档。虽然本章要探讨的许多概念都适用于所有二级存储设备，但是只考虑标准的磁盘驱动器是最简单的。

磁盘上的数据都存储在文件中，这是在电子媒介上组织数据的一种机制。所谓**文件**，就是相关数据的有名集合。从用户的角度来看，文件是可以写入二级存储设备的最小信息量。用文件组织所有信息，呈现出一个统一的信息存储视图。**文件系统**是操作系统提供的一个逻辑视图，使用户能够按照文件集合的方式管理数据。文件系统通常用**目录**组织文件。

> **文件**（file）：数据的有名集合，用于组织二级存储设备。
>
> **文件系统**（file system）：操作系统为它管理的文件提供的逻辑视图。
>
> **目录**（directory）：文件的有名分组。

文件是一个一般概念。不同类型的文件的管理方式不同。一般说来，文件存放的是（某种形式的）程序或（一种类型或另一种类型的）数据。有些文件的格式很严格，而有些文件的格式则很灵活。

可以把文件看作位序列、字节序列、行序列或记录序列。与存储在内存中的数据一样，要使文件中的位串有意义，必须给它们一个解释。文件的创建者决定了如何组织文件中的数据，文件的所有用户都必须理解这种组织方式。

11.1.1　文本文件和二进制文件

所有文件都可以被归为文本文件或二进制文件。在**文本文件**中，数据字节是ASCII或Unicode字符集中的字符（第3章介绍过字符集）。**二进制文件**要求基于文件中的信息给位串一个特定的解释。

> **文本文件**（text file）：包含字符的文件。
>
> **二进制文件**（binary file）：包含特定格式的数据的文件，要求给位串一个特定的解释。

术语文本文件和二进制文件会令人有些误解。听起来就像文本文件中的信息不是以二进制数据的形式存储的。计算机上的所有数据最终都是以二进制数字存储的。这些术语指的是格式化数位的方式，如8位或16位的位块将被解释为字符，另外还有其他专用的格式。

有些信息有字符表示法，通常使人更容易理解和修改。虽然文本文件只包括字符，但是这些字符可以表示各种各样的信息。例如，操作系统会将很多数据存储为文本文件，如用户账号的信息。用高级语言编写的程序也会被存储为文本文件，有时这种文件叫做源文件。用文本编辑器可以创建、查看和修改文本文件的内容，无论这个文本文件存储的是什么类型的信息。

而有些信息类型则是通过定义特定的二进制格式或解释来表示数据，以使其更有效且更符合逻辑。只有用专门解释这种类型的数据的程序才能够阅读或修改它。例如，存储图像信息的文件类型有很多，包括位图、GIF、JPEG和TIFF等。第3章中介绍过，即使它们存储的是同一个图像，它们存储信息的方式也不同。它们的内部格式是专有的，要查看或修改一种特定类型的二进制文件，必须编写专用的程序。这就是处理GIF图像的程序不能处理TIFF图像的原因。

有些文件你认为是文本文件，其实它并不是。例如，在字处理程序中输入并存储在硬盘中的报表。这个文档实际上被存储为一个二进制文件，因为除了文档中存储的字符外，它还包括有关格式、样式、边界线、字体、颜色和附件（如图形或剪贴画）的信息。有些数据（字符自身）被存储为文本，而其他信息则要求专用的格式。

11.1.2　文件类型

无论是文本文件还是二进制文件，大多数文件都包含有特定类型的信息。例如，一个文件可能包含有Java程序、JPEG图像或MP3音频片段。有些文件还可能包含有其他应用程序创建的内容，如Microsoft Word文档或Visio图片。文档中包含的信息的种类叫做**文件类型**。大多数操作系统都能识别一系列特定的文件类型。

说明文件类型的常用方法是将文件类型作为文件名的一部分。文件名通常由点号分为两部分，即主文件名和**文件扩展名**。文件扩展名说明了文件的类型。例如，文件名`MyProg.java`

中的扩展名.java说明这是一个Java源代码文件。文件名family.jpg中的扩展名.jpg说明这是一个JPEG图像文件。图11-1列出了一些常见的文件扩展名。

文件类型（file type）：文件（如Java程序或Microsoft文档）中存放的关于类型的信息。

文件扩展名（file extension）：文件名中说明文件类型的那部分。

根据文件类型，操作系统可以按照对文件有效的方式操作它。这样就大大简化了用户的操作。操作系统具有一个能识别的文件类型的清单，而且会把每种类型关联到特定的应用程序。在具有图形用户界面（GUI）的操作系统中，每种文件类型还有一个特定的图标。在文件夹中看到的文件都具有相应的图标。这使用户更容易识别一个文件，因为用户看到的不止是文件名，还有说明文件类型的图标。当双击这个图标后，操作系统会启动与这种类型的文件相关的程序以载入该文件。

例如，你可能想在开发Java程序时使用特定的编辑器，那么可以在操作系统中注册.java文件扩展名，并把它关联到要使用的编辑器。此后，每当要打开具有.java扩展名的文件时，操作系统都会运行这个编辑器。如何把文件扩展名和应用程序关联起来是由所采用的操作系统决定的。

扩展名	文件类型
txt	文本数据文件
mp3, au, wav	音频文件
gif, tiff, jpg	图像文件
doc, wp3	字处理文档
java, c, cpp	程序源文件

图11-1 常见的文件类型和它们的扩展名

有些文件扩展名是默认与特定的程序关联在一起的，如果需要，可以修改。某些情况下，一种文件类型能够关联到多种应用程序，因此你可以进行选择。例如，你的系统可能当前是把.gif文档与Web浏览器关联在一起的，所以只要一打开GIF图像文件，它就会显示在浏览器窗口中。你可以选择改变这种关联性，使得每当打开一个GIF文件，它就出现在你喜欢的图像编辑器中。

文件扩展名只说明了文件中存放的是什么。你可以任意命名文件（只要文件名中使用的字符在操作系统允许的范围之内）。例如，可以给任何文件使用.gif后缀，但这并不能使该文件成为一个GIF图像。改变文件扩展名不会改变文件中的数据或它的内部格式。如果要在专用的程序中打开一个扩展名错误的文件，只会得到错误信息。

11.1.3 文件操作

在操作系统协助下，可以对文件进行下列操作：

- 创建文件。
- 删除文件。
- 打开文件。
- 关闭文件。
- 从文件中读取数据。
- 把数据写入文件。
- 重定位文件中的当前文件指针。
- 把数据附加到文件结尾。
- 删减文件（删除它的内容）。
- 重命名文件。

• 复制文件。

让我们来分析一下每种操作是如何实现的。

操作系统用两种方式跟踪二级存储设备。它维护了一个表以说明哪些内存块是空的（也就是说可用的），还为每个目录维护了一个表，以记录该目录下的文件的信息。要创建一个文件，操作系统需要先在文件系统中为文件内容找一块可用空间，然后把该文件的条目加入正确的目录表中，记录文件的名字和位置。要删除一个文件，操作系统要声明该文件使用的空间现在是空的了，并要删除目录表中的相应条目。

大多数操作系统要求，在对文件执行读写操作前要先打开该文件。操作系统维护了一个记录当前打开的文件的小表，以避免每次执行一项操作都在大的文件系统中检索文件。当文件不再使用时，要关闭它，操作系统会删除打开的文件表中的相应条目。

无论何时，一个打开的文件都有一个当前文件指针（一个地址），说明下一次读写操作要发生在什么位置。有些系统还为文件分别设置了读指针和写指针。所谓读文件，是操作系统提交文件中从当前文件指针开始的信息的副本。发生读操作后，文件指针将被更新。写信息是把指定的信息记录到由当前文件指针所指的文件空间中，然后更新文件指针。通常，操作系统允许用户打开文件以便进行写操作或读操作，但不允许同时进行这两项操作。

打开的文件的当前指针可以被重定位到文件中的其他位置，以备下一次读或写操作。在文件结尾附加信息要求把文件指针重定位到文件的结尾，然后再写入相应的数据。

有时，删除文件中的信息是很有用的。所谓删减文件，是删除文件的内容，但不删除文件表中的管理条目。提供这项操作是为了避免删除一个文件，然后又重新创建它。有时，删减操作非常复杂，可以删除从当前文件指针到文件结尾的文件内容。

操作系统还提供了更改文件名的操作，叫做重命名文件。此外，操作系统还提供了创建一个文件内容的完整副本并给该副本一个新名字的功能。

11.1.4 文件访问

访问文件中信息的方式有很多。有些操作系统只提供一种文件访问类型，而有些操作系统则提供多种选择。文件的访问类型是在创建文件时设置的。

我们来分析两种主要的访问方法——顺序访问法和直接访问法。这两种访问法之间的区别就像第5章讨论过的磁带的顺序特性和磁盘的直接访问之间的区别。但是，任何类型的介质都可以存储这两种类型的文件。文件访问方法定义了重定位当前文件指针的方法。它们与存储文件的设备的物理限制无关。

最常用也是最容易实现的访问方法是**顺序访问法**，即把文件看作一种线性结构。这要求按顺序处理文件中的信息。读写操作将根据读写的数据量移动当前文件指针。有些系统允许把文件指针重置为文件的开头，还允许向前或向后越过几个记录。如图11-2所示。

采用**直接访问法**的文件会被概念性地划分为带编号的逻辑记录。直接访问法允许用户指定记录编号，从而把文件指针设置为某个特定的记录。因此，用户可以按照任何顺序读写记录，如图11-3所示。

顺序文件访问法（sequential file access）：以线性方式访问文件中的数据的方法。

直接文件访问法（direct file access）：通过指定记录编号直接访问文件中的数据的方法。

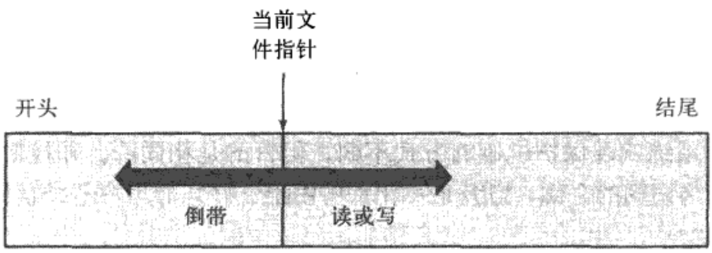


图11-2 顺序文件访问法

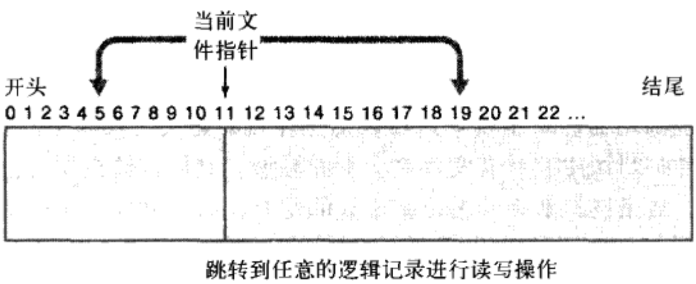


图11-3 直接文件访问法

直接文件访问法实现起来比较复杂，但在需要即刻使用大量数据（如数据库）的某个特定部分的情况下，这种方法很有用。

11.1.5 文件保护

在多用户系统中，文件保护的重要性居于首要地位。也就是说，除非是特许的，否则我们不想让一个用户访问另一个用户的文件。确保合法的文件访问是操作系统的责任。不同操作系统管理文件保护的方式不同。无论哪种情况，文件保护机制都决定了谁可以使用文件，以及为什么目的而使用文件。

例如，UNIX操作系统中的文件保护设置有三类，即Owner、Group和World。在每种类别下，你可以决定一个文件是可读的、可写的还是可执行的。采用这种机制，如果可以对一个文件进行写操作，就可以对它进行删除操作。

每个文件都由一个特定用户所有，通常是文件的创建者。Owner通常具有文件的最高访问许可。一个文件可能具有一个相关的组名。分组只是一个用户列表。一组中的用户都具有Group许可。例如，对于从事一个项目的所有用户可以这样分组。最后，访问系统的用户需要具有World许可。由于这些许可把访问权给予了最大数量的用户，所以它们通常是最受限制的。

采用这种方法，可以用3×3的表格说明文件具有的许可。

	读	写/删除	执 行
Owner	有	有	无
Group	有	无	无
World	无	无	无

假设这个表格表示Alpha项目使用的数据文件上的许可。文件的所有者（可能是项目经理）可以对它进行读写操作。假设所有者创建了一个分组TeamAlpha，包括项目组的所有成员，他

把这个分组与数据文件关联了起来。这个分组中的成员能够读文件中的数据，但是不能修改它。其他人都不能访问这个文件。注意，没有用户具有该文件的执行特权，因为它不是一个可执行的程序。

虽然其他操作系统实现保护机制的方式不同，但目的是相同的，即控制文件的访问，以防止蓄意获取不正当访问的企图，以及最小化那些由出于好意的用户不经意引起的问题。

11.2 目录

本章前面介绍过，目录是文件的有名集合。这是一种按照逻辑方式对文件分组的方法。例如，可以把某门课的笔记和试卷放在为这门课创建的目录下。操作系统必须仔细地跟踪目录和它们包含的文件。

大多数操作系统都用文件表示目录。目录文件存放的是关于目录中的其他文件的数据。对于任何指定的文件，目录中存放有文件名、文件类型、文件存储在硬盘上的地址以及文件的当前大小。此外，目录还存放文件的保护设置的信息，以及文件是何时创建的，何时被最后修改的。

建立目录文件的内部结构的方式有多种，这里我们不再详细介绍。不过，一旦有了目录文件，它就必须支持对目录文件的一般操作。例如，用户必须能列出目录中的所有文件。其他一般操作包括在目录中创建、删除或重命名文件。此外还有检索目录以查看一个特定的文件是否在目录中。

关于目录管理的另一个重要论题是如何反映目录中的文件关系，下一节将讨论这个问题。

RFID标签

假想一下，你在商店买了一包电池。在你离开商店时，这包电池“告诉”商店的售卖系统该补货了，因为电池存量很少了。射频识别技术（Radio-frequency identification，RFID）就使得这种情况成为可能。如果电池包装上有一个RFID标签，它就能告诉中央射频接收器自己在哪里。除了用于零售商店，RFID技术还用于跟踪货运集装架、图书馆的藏书、汽车和动物。研究员甚至还试验过在人体中植入RFID标签。

11.2.1 目录树

一个目录还可以包含另一个目录。包含其他目录的目录叫做*父目录*，被包含的目录叫做*子目录*。只要需要，就可以建立这种嵌套的目录来帮助组织文件系统。一个目录可以包含多个子目录。另外，子目录也可以有自己的子目录，这样形成了一种分级结构。因此，文件系统通常被看作**目录树**，展示了每个目录中的目录和文件。最高层的目录叫做**根目录**。

目录树（directory tree）：展示文件系统的嵌套目录组织的结构。

根目录（root directory）：包含其他所有目录的最高层目录。

例如，考虑图11-4所示的目录树。这个树表示文件系统的很小一部分，在使用Microsoft Windows操作系统的计算机上可以找到它。这个目录系统的根目录用驱动器符C:加\表示。

在这个目录树中，根目录包含三个子目录——`WINDOWS`、`My Documents`和`Program Files`。在`WINDOWS`目录中，有一个文件`calc.exe`和两个子目录——`Drivers`和`System`。

这些目录还包括其他的文件和子目录。记住，在真正的系统中，所有目录通常都包括更多的子目录和文件。

个人计算机通常使用文件夹来表示目录结构，这样构成了包容的思想（文件夹包含在另外的文件夹中，最终有些文件夹只包括文档或其他数据）。在使用图形化界面的操作系统中，使用图标来表示目录，通常是那种在真正的文件柜中摆放的马尼拉文件夹的图形。

注意，在图11-4中，有两个名为`util.zip`的文件（一个在`My Documents`中，一个在它的子目录`downloads`中）。嵌套的目录结构允许存在多个同名文件。任何目录下的所有文件的名字都必须是唯一的，但不同目录或子目录下的文件则可以是同名的。这些文件存放的数据可能相同，也可能不同，我们所知道的只是它们的名字相同。

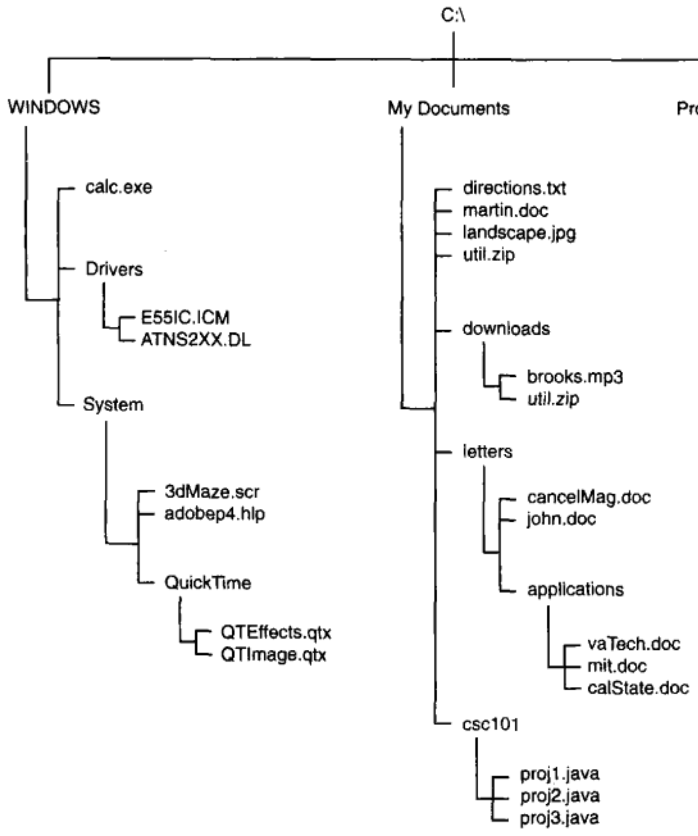


图11-4 Windows的目录树

无论何时，你都可以认为自己在文件系统中的某个特定位置（即特定的子目录）工作。这个子目录叫做当前**工作目录**。只要在文件系统中移动，当前工作目录就会改变。

工作目录（working directory）：当前活动的子目录。

图11-5中所示的目录树是UNIX文件系统的代表。比较图11-4和图11-5中的目录树。它们都展示了子目录包容的概念。不过，它们的文件和目录命名规则不同。UNIX是一个系统级的程序设计环境，因此使用了大量的缩写和代码作为目录和文件的名字。此外，还要注意，UNIX环境的根目录是用/表示的。

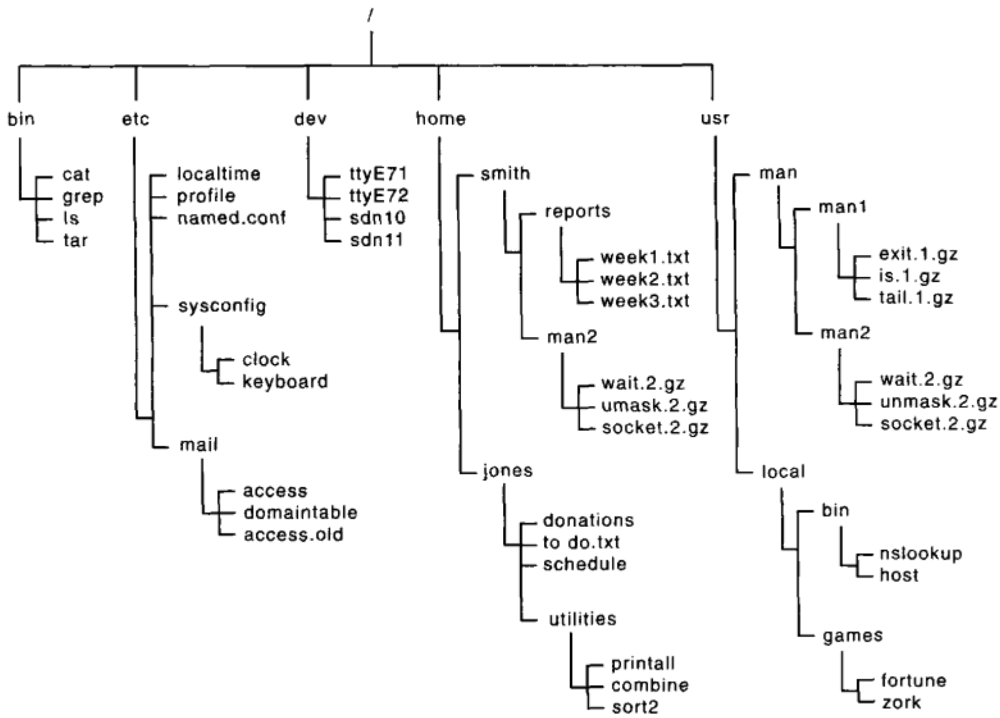


图11-5 UNIX的目录树

11.2.2 路径名

如何指定一个特定的文件或子目录呢？有下列几种方法。

如果使用的是具有图形化界面的操作系统，用鼠标双击目录，就可以打开它看到其中的内容。活动的目录窗口显示的是当前工作目录的内容。继续用鼠标点击，在文件系统中移动，改变当前的工作目录，直到找到你想要的文件或目录为止。

大多数操作系统只提供非图形化（基于文本）的界面，因此必须用文本说明文件的位置。对于存储在操作系统的批命令文件中的系统指令来说，这一点非常重要。像cd（表示改变目录）这样的命令，可以用文本模式改变当前的工作目录。

要用文本指定一个特定的文件，必须说明该文件的**路径**，即找到这个文件必须历经的一系列目录。路径可以是绝对的，也可以是相对的。**绝对路径**名从根目录开始，说明了沿着目录树前进的每一步，直到到达了想要的文件或目录。**相对路径**名则从当前工作目录开始。

> **路径**（path）：文件或子目录在文件系统中的位置的文本名称。
> **绝对路径**（absolute path）：从根目录开始，包括所有后继子目录的路径。
> **相对路径**（relative path）：从当前工作目录开始的路径。

让我们来看看每种类型的路径的实例。下面是一些图11-4所示的目录树中的绝对路径名：

```
C:\Program Files\MS office\WinWord.exe
C:\My Documents\letters\applications\vaTech.doc
C:\Windows\System\QuickTime
```

每个路径都从根目录开始，沿着目录结构向下推进。每个子目录都由\分隔。注意，一个

路径既可以说明一个特定的文档（如前两个例子），也可以说明整个子目录（如第三个例子）。

UNIX系统中的绝对路径也是这样的，只是分隔子目录的符号是/。下面是一些图11-5所示的目录树中的绝对路径名：

```
/bin/tar
/etc/sysconfig/clock
/usr/local/games/fortune
/home/smith/reports/week1.txt
```

相对路径是基于当前工作目录而言的。也就是说，它们是相对于当前位置的（因此而得名）。假设（图11-4中的）当前工作目录是C:\My Documents\letters。那么可以使用下列相对路径名：

```
cancelMag.doc
applications\calState.doc
```

第一个例子只说明了文件名，在当前工作目录中可以找到这个文件。第二个例子说明的是applications这个子目录中的文件。根据定义，任何有效的相对路径的第一部分都在工作目录中。

使用相对路径时，有时需要返回上层目录。注意，使用绝对路径不会遇到这种情况。大多数操作系统使用两个点（..）来表示父目录（一个点用于表示当前工作目录）。因此，如果工作目录是C:\My Documents\letters，下面的相对路径也是有效的：

```
..\landscape.jpg
..\csc101\proj2.java
..\..\WINDOWS\Drivers\E55IC-ICM
..\..\Program Files\WinZip
```

UNIX系统中的相对路径也是这样的。对于图11-5中的目录树，假设当前工作目录是/home/jones，下面的相对路径是有效的：

```
utilities/combine
../smith/reports
../../dev/ttyE71
../../usr/man/man1/ls.1.gz
```

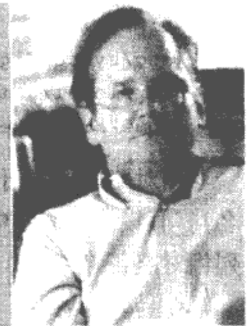
大多数操作系统允许用户（按照一定顺序）指定一组检索路径，以帮助解析对可执行程序的引用。通常用操作系统变量PATH指定这组路径，该变量存放的字符串中包含多个绝对路径。例如，假设（图11-5中的）用户jones有一套常用的工具程序，存储在目录/home/jones/utilities中。把这个路径添加到PATH变量中，它就成了检索jones要执行的程序的标准位置。因此，无论当前工作目录是什么，当jones执行printall程序时，将在他的工具目录中检索该文件。

舒缓软件

长期的压力可以导致心血管疾病、糖尿病、认知能力受损以及免疫功能低下等疾病。压力值的一个衡量标准是心率变化率（HRV），即心脏病专家在研究危险期病人时计量的两次心跳之间的毫秒数。对于一个健康的人来说，HRV值应该高，但要在一定的范围内。HeartMath公司提供了一套软件用于测量HRV值。它允许在标准的台式机上，使用耳夹或者手指夹获取用户的脉搏。在用户

集中精力呼吸或者思考积极的事物时测量他的心率几分钟，就可以使HRV达到目标范围。经过一天几次短暂的测试，就可以逐渐改变用户对压力的反应。

Lawrence Lessig



Lawrence Lessig是斯坦福法学院的法学教授，也是法学院的Internet和社会中心的创始人，曾经撰写过大量如何将法律（尤其是那些与版权和商标有关的法律）应用到Internet上的文章。他著名的言论是支持减少法律的约束，因为他认为在Internet领域过度使用版权法最终会抑制这种技术的潜在创造力。Lessig断言，所有创新都来自于过去的成就，而不可能是凭空得来的。因此，他认为一个自由的社会要进步，就不能过分控制过去的成就。

Lessig还认为计算机的特殊本性已经扩展了版权的概念，版权法管制的是副本，而副本在数字领域是无处不在的。对于与像书这样的实体有关的不可控制的动作（如在书边上加注释），在书被数字化后（如电子书）就变得可控制了，因为这样就要改变底层的代码，这是受版权保护的。

2001年，Lessig创建了非赢利性组织Creative Common，用于扩展那些能够合法供用户共享的原创作品的范围。此外，他还是Electronic Frontier Foundation的董事之一，这也是一个非赢利性的合法组织，专门支持公民自由，尤其是First Amendment保护的数字领域的言论自由。

11.3　磁盘调度

最重要的二级存储设备是磁盘驱动器。必须采用有效的方式才能访问存储在这些驱动器上的文件系统。实践证明，把数据传入或传出二级存储设备是一般的计算机系统的首要瓶颈。

第10章介绍过，CPU和主存的速度都比二级存储设备的数据传输速度快很多。这就是为什么一个进程要执行磁盘I/O操作，在等待信息传输的同时把使用CPU的机会让给另一个进程的原因。

由于二级I/O是一般计算机系统中最慢的部分，所以访问磁盘驱动器上的信息的方法对于文件系统至关重要。在计算机同时处理多个进程时，将建立一个访问磁盘的请求列表。操作系统用于决定先满足哪个请求的方法叫做**磁盘调度**。这一节将介绍几种磁盘调度算法。

磁盘调度（disk scheduling）：决定先满足哪个磁盘I/O请求的操作。

第5章介绍过，磁盘驱动器被组织得像一叠盘片，每个盘片被分为几个磁道，每个磁道又被分为几个扇区。所有盘片上对应的磁道构成了柱面。图11-6再现了第5章中使用过的磁盘驱动器图。

对我们的讨论来说，最重要的一点是在任意时刻都有一组读写头在所有盘片的特定柱面上盘旋。记住，*寻道时间*是读写头到达指定柱面所花费的时间。*等待时间*是盘片旋转到正确的位置以便能读写信息所花费的时间。在这两个时间中，寻道时间的要求更高，因此它是磁盘调度算法处理的重点。

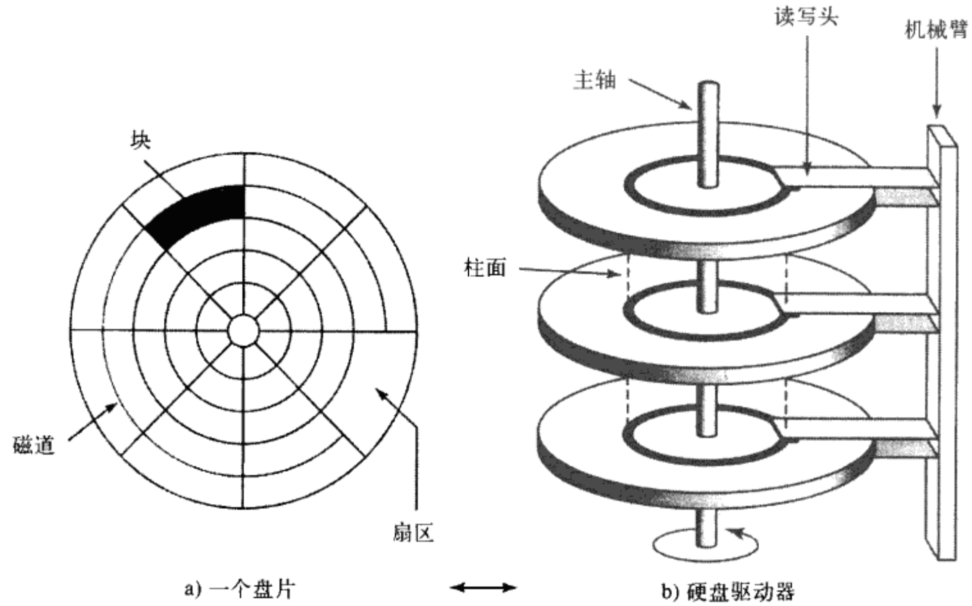


图11-6 磁盘驱动器

无论何时，磁盘驱动器可能都有一套必须满足的请求。从下面开始，我们只考虑请求引用的柱面（平行的同心圆）。为了简单起见，我们假定有100个柱面。假设某个特定时刻的柱面请求顺序如下：

49、91、22、61、7、62、33、35

此外假设读写头当前位于柱面26处。现在的问题是：磁头要移到哪个柱面？对于这个问题，不同的算法会生成不同的答案。

11.3.1 先到先服务磁盘调度法

第10章介绍过一种叫做*先到先服务*（FCFS）的CPU调度算法。类似的算法同样适用于磁盘调度。虽然它不是最有效的磁盘调度算法，但却是最容易实现的。

FCFS算法按照请求到达的顺序处理它们，并不考虑读写头的当前位置。因此，采用FCFS算法，读写头将从柱面26（它的当前位置）移到柱面49。满足了对柱面49的请求后（即读取或写入了信息），读写头将从柱面49移到柱面91。按照收到请求的顺序依此类推。

注意，在读写头从柱面91移到柱面22的过程中，要经过多个当前未解决的请求所要求的柱面。

11.3.2 最短寻道时间优先磁盘调度法

最短寻道时间优先（SSTF）磁盘调度算法将通过尽可能少的读写头移动满足所有未解决的请求。这种方法可能会在满足一个请求后改变读写头的移动方向。

让我们用这种算法来处理前面假设的状况。从柱面26开始，在所有未解决的请求中，与柱面26最近的是柱面22。因此，忽略请求到达的顺序，读写头将移动到柱面22以满足对它的请求。距离柱面22最近的被请求的柱面是柱面33，读写头将移到这里。距离柱面33最近的被请求的柱面是柱面35。现在距离最近的是柱面49，因此读写头下一步将移到这里。依此类推，余下被访问的柱面依次是49、61、62、91，最后是7。

虽然这种方法不能保证读写头的整体移动最少，但通常比FCFS算法有所提高。不过这种方法会引起一个重要问题。假设已有的请求还未解决，而新的请求仍然源源不断地到来，并且新的请求总是比早期的请求所需要的柱面离当前位置更近。那么从理论上来说，早期的请求将永远得不到满足，因为不断到来的请求总有优先权。这种情况叫做*饿死*。先到先服务磁盘调度算法不会出现饿死的情况。

11.3.3 SCAN磁盘调度法

在计算领域中，算法分析的经典例子是为使电梯到达有人等候的楼层而设计的方案。一般说来，电梯都是从一端移到另外一端（即从建筑的顶层到底层），为搭乘请求服务，然后再从底层上到顶层，为另外的请求服务。

SCAN磁盘调度算法的工作方式与之类似，只是在磁盘调度算法中没有上下移动，而是读写头向轴心移动，然后再向盘片边缘移动，就这样在轴心和盘片边缘之间来回移动。

让我们对前面的请求序列执行这个算法。与其他算法不同的是，我们要决定读写头最初移动的方向。假设它们是向编号较小的柱面移动的（当前位于柱面26处）。

在读写头从柱面26向柱面1移动的过程中，它们将满足对柱面22和7的请求。到达柱面1后，读写头将返回，向外移动到盘片边缘。在这个过程中，它们将按照下列顺序满足对柱面的请求，即33、35、49、61、62和91。

对新的请求没有任何特殊处理。它们可能在早期的请求之前受到服务，也可能在早期请求之后。这是由读写头当前的位置和它们移动的方向决定的。如果新的请求恰好在读写头到达柱面之前到达，它将被立刻处理。如果新的请求是在读写头刚经过那个柱面之后到达的，那么它必须等待读写头再次返回。不可能出现饿死现象，因为每个柱面都会被依次处理到。

这种算法的一些变体能用各种方法提高它的性能。例如，对盘片边缘柱面的请求可能需要等读写头从边缘移到轴心再从轴心移回到边缘。为了减少平均等待时间，*环形SCAN*算法把磁盘上的柱面序列看作环。也就是说，当读写头达到一端后直接返回另一端，之间不再处理请求。

另一种变体则是最小化到轴心和到盘片边缘的移动极限。读写头只移动到请求的最外面或最里面的柱面，不再移动到盘片边缘或轴心。在移动到下一个请求的柱面之前，有种算法会检查未处理的请求的列表，判断当前移到的方向是否正确。这种变体叫做LOOK磁盘调度算法，因为它会预先判断读写头是否应该继续按照当前的方向移动。

使老人留在家里

正在开发的许多新技术都使得老人在家独自生活变得更容易。一个例子是Health Buddy系统，它是一种远距离系统，可以远程管理关节炎、糖尿病和高血压这样的慢性疾病。Health Buddy是一种特制的计算机，可以根据近45项健康指标提供常规的建议。许多医疗传感器，例如测量糖尿病人的血糖水平的装置，都可以连接到这种计算机上。Health Buddy系统连接在电话上，每天把病人的数据传送给他们的医生。

小结

文件系统定义了组织二级存储设备的方式。文件是具有特殊内部结构的有名数据集合。

文本文件是字符流，二进制文件具有特定的格式，只有专用的应用程序才能识别。

用文件名的文件扩展名可以说明文件类型。操作系统具有可识别的文件类型的清单，以便能够用正确的应用程序打开它们，并且在图形化界面中显示正确的图标。文件扩展名可以与用户选择的任何应用程序关联在一起。

对文件执行的操作包括创建文件、删除文件、打开文件和关闭文件。当然，还要能够读写文件。操作系统为实现这些文件操作提供了办法。在多用户系统中，操作系统还要提供文件保护机制，以确保只有授权的用户才能访问文件。

目录用于组织磁盘上的文件。它们可以嵌套形成树形分级结构。路径名说明了特定文件或目录的位置，它们可以是绝对的，即从目录树的根开始，也可以是相对的，即从当前工作目录开始。

磁盘调度算法决定了处理未解决的请求的顺序。先到先服务磁盘调度算法是顺序处理请求，不过这种方法效率不高。最短寻道时间优先调度法更有效一些，但却会产生饿死现象。SCAN调度算法采用的策略与电梯采用的一样，即从磁盘的一端向另一端检索。

道德问题：垃圾邮件

每天收电子邮件时，人们都会看到许多不想见到的消息。由于广告业的发展，营销人员一直在探索与潜在客户的新联系方法。谁的邮箱中没有被传单、产品目录和“特殊邀请”塞满过？既然这么多人使用Internet和电子邮件，市场商人当然会利用这种更廉价的方式来联系更多的客户。

垃圾邮件（不受欢迎的电子邮件的绰号）曾经一度只是一种骚扰。不过随着它的数量日益增长，人们对它的关注也日益增加。美国商务的经济支出增长得非常快。与家用邮箱不同的是，传送垃圾邮件需要支付给负责通信的ISP一定费用，而终端用户要为ISP提供的上网账户支付费用。再加上人们处理垃圾邮件所花费的时间猛涨。最新的估计指出，美国商务每年要为垃圾邮件支付约220亿美元，每个员工每年约要花费2000美元。当前垃圾邮件占电子邮件的83%，根据专家预言，这一数字将继续上涨。这么高的花费，只会增加许多公司的负担。

虽然美国的垃圾邮件问题是最严重的，但是许多其他国家也都在为此斗争。在加拿大，超过60%的邮件是垃圾邮件；在英国，将近52%的邮件是垃圾邮件；在德国，这个比例是41%左右。

营销人员声称，任何控制垃圾邮件的行为都限制了言论自由。他们还声称，垃圾邮件只是资本原则的另一种形式，限制公司使用让客户知道他们的产品和服务的手段没有任何好处。分发广告者声称客户有权知道产品和服务信息，他们以此来支持自己的立场。如果不主动邮寄大量的广告，许多公司都不知道该如何找到客户。

几年来，立法者一直不愿意参与这种冲突。他们倾向于同意营销人员的观点，即垃圾邮件虽然是大多数人不想看到的，但它是受保护的活动，是合理的商业策略。由于市民和公司都开销很大，所以这种观点开始改变了。2004年1月，颁布了CAN-SPAM法案（Controling the Assault of Non-Solicited Pornography and Marketing）。这项法案的目的是保护消费者不受商业电子邮件的打扰。该法案规定，现在所有电子邮件都要有精确的路由信息，主题栏要精确地反映出消息的内容，还要有决定选项，接收者可以用来取消未来的邮件。此外，这里消息必须明确标识为广告，还要具有发送者的有效邮寄地址。垃圾邮件的发送者必须在10个工作日内响应任何请求。违反这一法案，将被罚款11 000美元。除了这项新法案，各个州还有权强化自己的垃圾邮件管理的法规。虽然CAN-SPAM法案不能完全杜绝垃圾邮件的传播，但立法者希望它能够有助于消费者减少这方面的开支。

练习

判断练习1～15中的陈述的对错：

A. 对　　B. 错

1. 文本文件存储的二进制数据是按照8位或16位的分组组织的，这些分组被解释为字符。
2. 用高级语言编写的程序是存储为文本文件的，也叫做源文件。
3. 文件类型决定了能够对文件执行哪些操作。
4. 当前文件指针指的是文件的结尾。
5. 顺序访问法和直接访问法获取数据所花的时间量相同。
6. 有些操作系统为文件分别维护有读指针和写指针。
7. UNIX文件许可允许一组用户以各种方式访问一个文件。
8. 大多数操作系统用文件表示目录。
9. 在目录系统中，如果两个文件处于不同的目录下，那么它们可以具有相同的名字。
10. 相对路径是相对于目录分级结构的根而言的。
11. 绝对路径和相对路径总是等长的。
12. 操作系统要负责管理对磁盘驱动器的访问。
13. 寻道时间是磁盘的读写头到达特定的柱面所花费的时间。
14. 最短寻道时间优先磁盘调度算法是尽可能少地移动读写头以满足未解决的请求。
15. 先到先服务磁盘调度算法是尽可能少地移动读写头以满足未解决的请求。

为练习16～20中的文件找到匹配的文件扩展名。

A. txt　　B. mp3、au和wav

C. gif、tiff、jpg　　D. doc和wp3

E. java、c和cpp

16. 音频文件。
17. 图像文件。
18. 文本数据文件。
19. 程序源文件。
20. 字处理文件。

为练习21～23中描述的用途找到匹配的符号。

A. /　　B. \

C. ..

21. 在Windows环境中用于分隔路径中的目录名的符号。
22. 在UNIX环境中用于分隔路径中的目录名的符号。
23. 在相对路径中用于表示父目录的符号。

练习24～57是问答题或简答题。

24. 什么是文件？
25. 请区分文件和目录。
26. 请区分文件和文件系统。
27. 为什么文件是一般概念，而不是技术概念？
28. 请列举并说明两种基本的文件分类。
29. 为什么说术语“二进制文件”用词不当？
30. 请区分文件类型和文件扩展名。
31. 如果把一个文本文件命名为`myFile.jpg`，会发生什么情况？
32. 操作系统如何利用它识别出的文件类型？
33. 操作系统是如何跟踪二级存储设备的？
34. 打开和关闭文件是什么意思？
35. 删减文件是什么意思？
36. 请比较顺序文件访问法和直接文件访问法。
37. 文件访问是独立于物理介质的。
    a) 如何实现磁盘的顺序访问？
    b) 如何实现磁带的直接访问？
38. 什么是文件保护机制？
39. UNIX是如何实现文件保护机制的？
40. 根据下列文件许可，回答后面的问题。

	读	写或删除	执　行
Owner	有	有	有
Group	有	有	无
World	有	无	无

    a) 谁可以读文件？
    b) 谁可以写或删除文件？
    c) 谁可以执行文件？
    d) 你对文件内容有何了解？
41. 目录必须存放的关于每个文件的最少信息是什么？
42. 大多数操作系统如何表示目录？
43. 回答下列有关目录的问题。
    a) 包含另一个目录的目录叫什么？

b) 被包含在另一个目录中的目录叫什么？

c) 不包含在任何目录中的目录叫什么？

d) 展示了目录的嵌套组织形式的结构叫什么？

e) 把(d)中的结构与第9章介绍的二叉树结构联系起来。

44. 无论何时，你正在使用的目录叫做什么？

45. 什么是路径？

46. 请区分绝对路径和相对路径。

47. 根据图11-4所示的目录树，说明下列文件或目录的绝对路径。

a) `QTEffects.qtx`

b) `brooks.mp3`

c) `Program Files`

d) `3dMaze.scr`

e) `Powerpnt.exe`

48. 根据图11-5所示的目录树，说明下列文件或目录的绝对路径。

a) `tar`

b) `access.old`

c) `named.conf`

d) `smith`

e) `week3.txt`

f) `printall`

49. 假设当前工作目录是`C:\WINDOWS\System`，根据图11-4所示的目录树，说明下列文件或目录的相对路径。

a) `QTImage.qtx`

b) `calc.exe`

c) `letters`

d) `proj3.java`

e) `adobep4.hlp`

f) `WinWord.exe`

50. 根据图11-5所示的目录树，说明下列文件或目录的相对路径。

a) 当工作目录是根目录时`localtime`的相对路径。

b) 当工作目录是`etc`时`localtime`的相对路径。

c) 当工作目录是`utilities`时`printall`的相对路径。

d) 当工作目录是`man2`时`week1.txt`的相对路径。

51. 计算机系统的主要瓶颈是什么？

52. 为什么磁盘调度法注重的是柱面，而不是磁道和扇区？

53. 请列举并说明三种磁盘调度算法。

练习54～56需要使用下列柱面请求列表。这里列出的是它们的接收顺序。

40、12、22、66、67、33、80

54. 如果采用FCFS算法，请列出处理请求的顺序。假设磁盘当前定位在柱面50。

55. 如果采用SSTF算法，请列出处理请求的顺序。假设磁盘当前定位在柱面50。

56. 如果采用SCAN算法，请列出处理请求的顺序。假设磁盘当前定位在柱面50，读写头向大编号的柱面移动。

57. 请解释饿死的概念。

思考题

1. 计算领域充斥着文件的概念。如果没有存储文件的二级存储设备，计算机还有用吗？

2. 本章介绍的磁盘调度算法听起来很熟悉。我们在什么环境中讨论过类似的算法？这些算法有哪些相似之处？又有哪些不同？

3. 文件和目录以及文件夹和档案柜之间有什么相似性吗？显然“文件”这个名字来自这些概念。那么对于“文件”来说，哪些地方具有上述相似性，哪些地方没有呢？

4. Internet上的垃圾邮件就像推销电话一样。目前美国有法律允许电话用户请求从推销员的电话列表中删除自己的名字。那么是否应该对垃圾邮件建立相似的法律保障呢？

5. 以你的观点，带兜售信息的垃圾邮件是合理的商业策略，还是一种电子骚扰？为什么？

6. 许多CAN-SPAM法案的批评者认为该法案很大程度上是无效的。你认为政府应该在根除垃圾邮件方面做得更严厉一些吗？或者你认为这与First Amendment给消费者提供的言论自由相冲突吗？

第六部分 应用程序层

第12章 信息系统

大多数人都在应用程序层与计算机打交道。也就是说，即使一个用户对应用程序层之下的各个计算层一无所知，也可以使用应用软件。关于这一层，我们的目标是让你了解各种应用系统是如何运作的。划分应用软件的方式多种多样。本章的重点是一般的信息系统。第13章将讨论人工智能领域的应用，第14章的重点是模拟、图形学和嵌入式系统。

计算机是用来管理和分析数据的。当今，计算机的效应在我们的生活中几乎无处不在。我们用一般信息系统来管理所有数据，从运动统计数字到薪水册的数据，无所不包。同样地，收银机和ATM都有大型的信息系统支持。本章将分析一些多功能软件，特别是电子制表软件和数据库管理系统，它们将有助于我们组织和分析大量的数据。此外，我们还会分析把信息安全地保存在系统中的固有问题。

目标

学完本章之后，你应该能够：

- 定义一般信息系统的角色。
- 解释电子数据表的结构。
- 为数据的基本分析创建电子数据表。
- 用内置函数定义适用的电子数据表公式。
- 设计可扩展的、灵活的电子数据表。
- 描述数据库管理系统的元素。
- 描述关系数据库的结构。
- 在数据库的各元素间建立关系。
- 编写基本的SQL语句。
- 描述实体-关系图。
- 讨论CIA三元组。
- 描述密码学在数据保护中的作用。

12.1 信息管理

本书不止一次地把数据定义为原始事实，信息表示组织起来帮助我们回答问题以及解决问题的数据。**信息系统**一般被定义为帮助我们组织和分析数据的软件。

> **信息系统**（information system）：帮助我们组织和分析数据的软件。

任何应用程序都是管理数据的，不过有些程序则采用特定的结构以特定的方式管理数据。还有一些专用应用程序则使用特定的技术解决问题。例如，下一章将介绍为支持人工智能这个计算领域需要的分析而提供的各种组织数据的方式。

然而，大多数情况是一般性的，它们不需要特别考虑。我们只需要管理数据，捕捉数据间的关系。这种情况不需要任何特别的组织和处理。它们需要的是灵活的软件工具，能够让用户指示和管理数据的组织，具备用多种方式分析数据的能力。

两种最常用的一般信息系统是*电子制表软件*和*数据库管理系统*。电子制表软件是一种很方便的工具，它采用可扩展的公式定义数据间的关系，适用于基本的数据分析。数据库管理系统适用于需要经常检索并且有组织的大量数据。

有很多关于电子制表软件的安装与用法的书。关于数据库管理系统的书也不少。本章的目的不是详细探讨这两种信息系统，而在于介绍它们的用途和多功能性。学完本章之后，你应该能够创建这些系统的基础版本，而且为详细研究它们打下了基础。

Ellis Island数据库

自从2001年4月17日开放以来，Ellis Island网站（www.ellisisland.org）已经有了60亿的点击量。该网站的可检索数据库中存放了到达美国的2500万乘客的姓名、年龄和原始国籍，甚至包括他们抵达时乘坐的轮船。Ellis Island移民局记录了从1892～1924年的所有移民和访客的信息，这一站点包括一些著名的到达者，如Rudyard Kipling（1892）、Sigmund Freud（1909）、Harry Houdini（1914）和Albert Einstein（1921）。

12.2　电子制表软件

目前可用的电子制表软件多种多样。即使你没有什么背景知识，你很可能已经使用过电子制表软件。虽然每种电子制表软件的功能和语法都有细微差别，不过它们依赖的基本概念是相同的。本章讨论的重点在于这些通用的概念。我们使用的实例采用的是Microsoft Excel电子制表软件的语法和功能。

所谓**电子制表软件**，是一种软件应用程序，它允许用户用带标签的**单元格**组织和分析数据。单元格可以存放数据或用于计算值的公式。存储在单元格中的数据既可以是文本，也可以是数字或其他特殊数据（如日期）。

电子制表软件（spreadsheet）：允许用户用单元格组织和分析数据的程序。
单元格（cell）：电子数据表中用于存放数据或公式的元素。

如图12-1所示，可以用行列标号引用电子数据表的单元格，通常用字母指定列，用数字指定行。因此，可以用诸如A1、C7和G45这样的标号来引用单元格。对于第26列之后的列，电子制表软件用两个字母作为列标号，所以，有些单元格的标号是AA19。通常，电子数据表有一个合理的最大行数，如256。另外，大多数电子制表软件会把多个表组合在一个大的交互系统中。

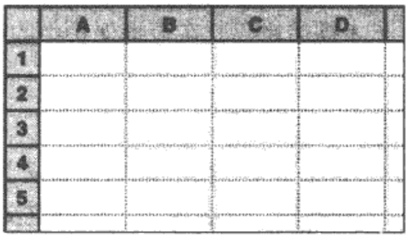


图12-1　由带标签的单元格构成的电子数据表

许多情况下都会用到电子制表软件，它们常常要管

理大量的数值和计算。我们来看一个小型的实例，以说明电子数据表的基本原理。假设我们搜集了几周以来向一组辅导教师求助的学生的数据。我们掌握了5周以来每周分别向三位辅导教师（Hal、Amy和Frank）求助的学生的人数。现在我们想对这些数据执行一些基本的分析，可以得到图12-2所示的电子数据表。

	A	B	C	D	E	F	G	H
1								
2				Tutor				
3			Hal	Amy	Frank	Total	Avg	
4		1	12	10	13	35	11.67	
5		2	14	16	16	46	15.33	
6	Week	3	10	18	13	41	13.67	
7		4	8	21	18	47	15.67	
8		5	15	18	12	45	15.00	
9		Total	59	83	72	214	71.33	
10		Avg	11.80	16.60	14.40	42.80	14.27	
11								
12								

图12-2 包含数据和计算的电子数据表

这个电子数据表除了具有其他数据外，还包括原始数据。例如，单元格C4存放的是Hal在第1周辅导过的学生数。从C4到C8，存放的是Hal在这5周中每周辅导的学生数。同样地，Amy辅导的学生人数存放在D4到D8中，Frank辅导的学生人数存放在E4到E8中。可以把一行中的数据看作是意义相同的。在上面的例子中，每行中的数据是在任意指定的一周中辅导教师辅导过的学生人数。

在单元格C9、D9和E9中，电子制表软件计算并显示出了每位辅导教师在5周中帮助过的学生的总数。在单元格C10、D10和E10中，还计算并显示了每位辅导教师平均每周帮助的学生人次。同样地，从F4到F8显示了每周受到（所有辅导老师）帮助的学生的总数。从G4到G8是每周每位老师辅导的学生的平均数。

除了计算每位老师每周辅导的学生的总数和平均数外，电子制表软件还能够计算其他的统计值。单元格F9是所有老师在5周中一共辅导过的学生人次。F10是所有老师平均每周辅导的学生人次，G9是每位老师5周中平均辅导的学生人次。最后，G10是每位老师平均每周辅导的学生人次。

第A列和B列中的数据以及第2行和第3行中的数据只是用作标签，说明了其余单元格中存储的是什么值。这些标签只是为了便于人们理解，与计算无关。

注意，图12-2使用了不同颜色表示标签和数值（由于这里是黑白图，所以显示不出来）。大多数电子制表软件允许用户控制单元格中的数据的外观和格式。用户可以设置数据的字体、样式、颜色和对齐方式（如居中或左对齐）。对于实数值（如上面例子中的平均数），可以设置显示多少位小数。在大多数电子制表软件中，用户还能够设置是否显示网格线（这个例子显示了网格线）、背景颜色或单元格的图案。用电子制表软件中的菜单选项或按钮可以设置这些用户首选项。

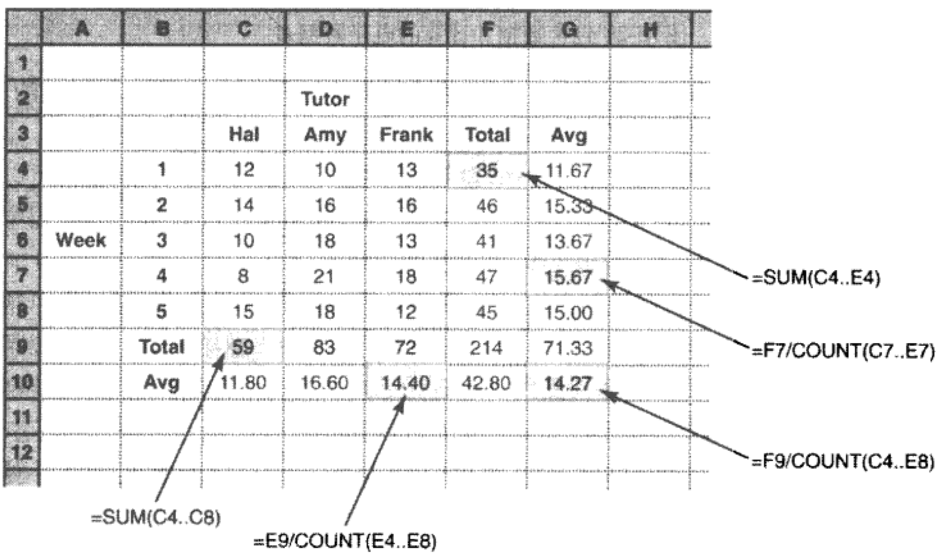
12.2.1　电子数据表公式

前面的例子中执行的几个运算使我们对辅导情况有了个全局了解。这个例子说明执行这些运算相对来说是比较简单的。你可能会说，用计算器也能很快得到同样的统计数字。不错，不过电子数据表的好处在于易于修改和易于扩展。

如果已经正确建立了电子数据表，那么就可以添加辅导老师，添加更多周的数据，或者更改已经存在的数据，对应的计算结果会自动更新。例如，尽管我们的例子中只有三位老师的数据，但这个表还可以处理几百位老师的数据。此外，它还可以轻松地处理一年的数据，而不止是5周的。

电子数据表的这种能力源于我们创建并存储在单元格中的公式。图12-2的例子中的所有总值和平均数都是用公式计算的。把公式存储在一个单元格中，这个单元格就会显示该公式的结果。因此，在查看电子数据表中的值时，很难分辨出单元格中的数据是直接输入的，还是通过公式计算出的。

图12-3展示的电子数据表与图12-2中的相同，只是标示出了存放公式的单元格。这个例子中的公式（和许多电子制表软件一样）都是以等号（=）开头的。电子数据表就是通过这一点知道哪些单元格存放的是要计算的公式。



	A	B	C	D	E	F	G	H
1								
2				Tutor				
3			Hal	Amy	Frank	Total	Avg	
4		1	12	10	13	35	11.67	
5		2	14	16	16	46	15.33	
6	Week	3	10	18	13	41	13.67	
7		4	8	21	18	47	15.67	
8		5	15	18	12	45	15.00	
9		Total	59	83	72	214	71.33	
10		Avg	11.80	16.60	14.40	42.80	14.27	
11								
12								

图12-3　一些单元格中的公式

这个例子中的公式（通过列标号和行标号）引用了特定的单元格。在计算公式时，将用所引用的单元格中的值计算结果。每当电子数据表有变化，其中的公式都会被重新计算，所以表中的数据总是最新的。电子数据表是动态的，能对变化立即作出响应。如果改变第2周中Frank辅导的学生数，那么使用这个值的总值和平均数都会被立刻重算，以反映修改过的数据。

电子数据表中的公式可以利用使用标准符号（+、-、*和/）的基本数学运算，还可以利用软件内置的**电子数据表函数**。在前面的例子中，单元格C9使用了SUM函数来计算C4、C5、C6、C7和C8的和。

> **电子数据表函数**（spreadsheet function）：电子制表软件提供的可用于公式的计算函数。

由于函数通常作用于一系列连续的单元格，所以电子制表软件提供了一种便捷的方式，即指定单元格的**范围**。从语法上来讲，范围是由两个圆点及两端加两个单元格标号构成的。一个范围可以是一行中的一组单元格，如C4..E4，也可以是一列中的一组单元格，如C4..C8。此外，范围还可以是一个矩形块，指定了左上角的单元格标号和右下角的单元格标号。例如，C4..E8中包括单元格C4到C8、D4到D8和E4到E8。

> **范围**（range）：用端点指定的一组连续单元格。

图12-3中所示的几个公式使用了COUNT函数，用于计算指定范围内的非空单元格数。例如，单元格G7中的公式是用单元格F7中的值除以范围C7..E7中的非空单元格数（即3）。

G7中的公式可以改写为下列形式：

=SUM(C7..E7)/3

根据电子数据表的当前状态，这个公式计算的结果应该与原来的公式相同。不过，这个公式没有原来的公式那么好，原因有两点。其一，C7到E7的和已经计算出来了（存在F7中），所以没必要再计算一次。任何数据变化都会影响F7的值，从而会影响G7的值。电子数据表应该将所有这种关系考虑在内。

其二（更重要），除非特别适合，否则要尽量避免在公式中使用常量。在本例子中，使用3作为预定的辅导老师人数就限制了添加或删除辅导老师的能力。电子数据表中的原始数据发生变化，其中公式的值就会变化，公式自身也应该对插入和删除操作作出类似的响应。如果我们插入了另一个辅导老师的一列数据，那么F和G列中的原始公式的范围将会自动更新，以反映这种变化。例如，如果插入了一个新的辅导老师的数据列，那么单元格F4中的公式会自动转移到单元格G4，现在的公式是：

=SUM(C4..F4)

也就是说，单元格的范围会自动扩展，以加入新插入的数据。同样地，其他函数中的COUNT函数使用的范围也会改变，生成一个新的正确平均数。如果在G7的公式中使用常量3，那么插入新的列后，计算结果就不正确了。

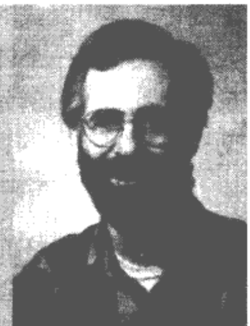
电子制表软件通常会提供大量的函数。一些函数执行的是数学或统计运算、一般的金融计算或者是文本或日期的特殊运算。另一些函数则允许用户建立单元格间的逻辑关系。图12-4列出了一些常见的电子数据表函数。典型的电子制表软件会提供许多这样的函数，以便用户在公式中使用。

电子数据表的另一灵活之处是能够整行或整列地复制值或公式。复制公式时，单元格间的关系都将维持不变，因此很容易设置一整套类似的计算。例如，在上面的例子中，要在单元格F4到F8中输入总值的计算公式，只需要在F4中输入这个公式，然后把它复制到整个列即可。在复制的公式中，对单元格的引用会被自动更新，以反映新的公式所在的行。由于我们的例子比较小，只记录了5周的数据，所以复制操作不会节省太多操作。但请想象一下，如果我们记录了整年的数据，要创建52个求和公式，制表软件的复制功能只需要一个操作就可以完成了。

函　数	计　算
SUM(val1, val2, …) SUM(range)	指定的一组值的和
COUNT(val1, val2, …) COUNT(range)	非空单元格的个数
MAX(val1, val2, …) MAX(range)	指定的一组值中的最大值
SIN(angle)	指定角度的正弦值
PI()	π的值
STDEV(val1, val2, …) STDEV(range)	指定的采样值的标准差
TODAY()	今天的日期
LEFT(text, num_chars)	指定文本的最左边的字符
IF(test, true_val, false_val)	如果test是true，则返回true_val，否则返回false_val
ISBLANK(value)	如果指定的值引用的是一个空单元格，则返回true

图12-4　一些常用的电子数据表函数

Daniel Bricklin



本书中的许多传记介绍的都是计算机界的最高奖ACM图灵奖的获得者。除了图灵奖，ACM还为35岁以下的年轻人设立了Grace Murray Hopper奖，以奖励他们的杰出贡献。这个奖项的要求是：

奖给本年度杰出的年轻计算机工作者……选拔的唯一标准是近来的学术贡献……候选人在做出具有候选资格的贡献时年龄不能超过35岁。

Daniel Bricklin赢得了1981年的Hopper奖，他的获奖词如下：

为了他对个人计算，尤其是VisiCalc的设计做出的贡献而授予他这个奖。Bricklin在开发Visual Calculator时不遗余力，正体现了ACM要通过这个颁奖活动而维持的优秀和优雅的科学研究品质。

Daniel Bricklin生于1951年，是计算机时代的一员。他的大学生涯开始于1969年，在麻省理工学院就读数学专业，不过很快就转读了计算机科学。他曾经在MIT的计算机科学实验室工作过，从事交互式系统的开发，并在此遇到了以后的商业合作伙伴Bob Frankston。毕业后，他受雇于Digital Equipment公司，致力开发计算机化的排版系统，协助设计WPS-8字处理产品。

1977年，在结束了FasFax公司（一家收银机制造商）的短期工作后，Bricklin报读了哈佛商学院的MBA课程。在此期间，他开始构思一种能够像字处理器操作文本一样操作数字的程序。这种程序会对商业界带来巨大的冲击。他和昔日MIT的搭档Bob Frankston联手把这个梦想变成了现实。Bricklin负责设计，Frankston负责编程，第一个电子制表软件VisiCalc就这样诞生了。1978年，他们成立了Software Arts公司，开始生产VisiCalc，

并把它推向市场。1979年秋制造出了适用于Apple II的版本，每个拷贝售价100美元。1981年制造出了适用于IBM PC的版本。

Bricklin相信软件不应该私有化，所以决定不为VisiCalc申请专利。尽管没有专利，但是这家公司在4年间雇员已经达到了125人。不过，一家新公司Lotus的出现，对VisiCalc的销售造成了严重的冲击，他们发布的制表软件包Lotus 1-2-3，功能更强大，界面也更友好。经过Software Arts和VisiCorp之间漫长而昂贵的法庭之争后，Bricklin被迫将公司出售给Lotus Software。此后，Microsoft公司的Excel胜过了Lotus 1-2-3。无论Lotus 1-2-3还是Excel，都是以VisiCalc为基础的。

在Lotus Software短期地担任顾问后，Bricklin再次组建了一个新公司Software Garden。作为这家公司的总裁，他开发了以软件为模型模拟软件的其他部分的程序，为此他获得了1986年Software Publishers Association颁发的最佳程序设计工具奖。1990年，他创立了Slate公司，为笔输入计算机开发应用软件，这是一种小型计算机，使用手写笔代替键盘输入。成立4年后，Bricklin结束了Slate，返回了Software Garden。

1995年，他成立了Trellix公司，一家私人站点发布技术的领先提供商。他把网络比喻成机动车早期的原始小路，当时没人预见到有一天会出现这么庞大的高速公路系统。"我们需要理解的不是那么多技术"，他解释道，"而是技术的发展以及它能创造什么。像电流和电话一样，电子商务使我们可以用技术来实现我们在做的事，而且做得更好。"

12.2.2 循环引用

注意，电子数据表的公式中可以有**循环引用**，这种引用是不可能解决的，因为一个公式的结果始终是基于另一个公式的，反之亦然。例如，如果单元格B15中的公式如下：

=D22+D23

而单元格D22中的公式是：

=B15+B16

这就是一个循环引用。B15的结果要使用D22的值，而D22的结果又是由B15决定的。

循环引用通常不会这么明显，可能会涉及多个单元格。图12-5展示了一个更复杂的情况。最终，单元格A1的结果是由D13决定的，反之亦然。电子制表软件通常能探测出这些问题并提示错误信息。

单元格	内　容
A1	=B7*COUNT(F8..K8)
B7	=A14+SUM(E40..E50)
E45	=G18+G19 – D13
D13	=D12/A1

图12-5　不能解决的循环引用

> **循环引用**（circular reference）：在计算结果时要错误地彼此依赖的一组公式。

12.2.3 电子数据表分析

电子数据表之所以有用，原因之一是它们具有多功能性。电子数据表的用户可以决定其中的数据表示什么以及数据间的关系。因此，电子数据表分析可以应用于任何领域。例如，我们可以用电子数据表来：

- 跟踪销售情况

- 分析运动统计数字
- 维护学生的成绩单
- 保存汽车的维修记录
- 记录和总结旅行开销
- 跟踪项目活动和日常安排
- 计划股票购买

事实上，潜在的应用是无限多的。一般说来，电子数据表的运算在商业领域的大量特定情况中是不可或缺的。如果没有电子制表软件，情形之糟糕可能会令你吃惊不已。

电子数据表的动态特性也使得它极其有用。一旦正确建立了电子数据表公式，那么计算会将数据的更改、添加或删除自动考虑在内。

电子数据表的动态特性还提供了进行**模拟假设分析**的功能。我们可以在电子数据表中设置一些假设，然后通过改变适当的值来质疑这些假设。

> **模拟假设分析**（what-if analysis）：修改电子数据表中表示假设的值，以观察假设的变化对相关数据有什么影响。

例如，假设我们创建了一个电子数据表用于估计举办一个研讨会的花费和潜在利润。我们可以输入参加者的人数、门票价格、资料费、会议室租金以及其他可能影响最终结果的数据，然后问自己假设分析的问题，看看随着各种条件的变化会出现哪些情况：

- 如果参加者人数减少了10%将会怎么样？
- 如果门票价格增加了5美元将会怎么样？
- 如果把资料费减少一半将会怎么样？

在问这些问题的同时改变相应的数据。如果已经正确建立了所有公式间的关系，那么每个改变都会立刻展示给我们其他数据发生了哪些变化。

商业分析师以各种方式标准化了这一过程，电子制表软件成了他们日常工作的主要工具。成本效益分析、收支平衡计算以及预计销售出价都是通过组织电子数据表中的数据和公式来考虑适当的关系。

12.3　数据库管理系统

几乎所有复杂的数据管理情况都要依靠下层的数据库和允许用户（人或程序）与之交互的支持结构。**数据库**可以简单定义为结构化的数据集合。**数据库管理系统**（DBMS）是一组软件和数据的组合，由下列几部分构成：

- 物理数据库——存放数据的文件的集合。
- 数据库引擎——支持对数据库内容的访问和修改的软件。
- 数据库模式——存储在数据库中的数据的逻辑结构的规约。

> **数据库**（database）：结构化的数据集合。
>
> **数据库管理系统**（database management system）：由物理数据库、数据库引擎和数据库模式构成的软件和数据的组合。

数据库引擎将与专用的数据库语言交互，这种语言允许用户指定数据的结构，添加、修改和删除数据，以及**查询**数据库以获取指定的数据。

数据库**模式**提供了数据库中的数据的逻辑视图，独立于数据的物理存储方式。假设以一种有效的方式实现了数据库的物理结构，那么从数据库用户的观点来看，逻辑模式是更加重要的数据库视图，因为它展示了数据项之间的关系。

图12-6展示了数据库管理系统的各个组件之间的关系。用户将先与数据库引擎软件交互，决定或修改数据库的模式。然后再与数据库引擎交互，访问和修改存储在硬盘上的数据库的内容。

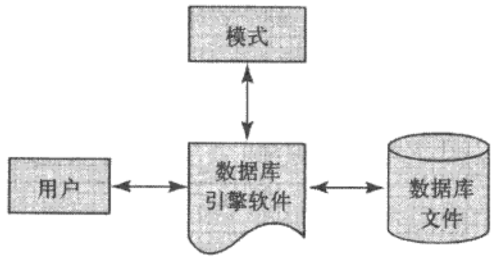


图12-6 数据库管理系统的组件

> **查询**（query）：提交给数据库的信息请求。
> **模式**（schema）：数据库中的数据的逻辑结构的规约。

12.3.1 关系模型

流行的数据库管理模型有几种，但是多年来占统治地位的还是**关系模型**。在关系DBMS中，用**表**组织数据项和它们之间的关系。表是**记录**的集合。记录是**域**的集合。数据库表的每个域都包括一个数值。表中的每个记录都包含相同的域。

> **关系模型**（relational model）：用表组织数据和数据之间的关系的数据库模型。
> **表**（table）：数据库记录的集合。
> **记录**（**或对象**、**实体**）（record(or object, entity)）：构成一个数据库实体的相关的域的集合。
> **域**（**或属性**）（field(or attribute)）：数据库记录中的一个值。

数据库表中的记录又叫*数据库对象*或*实体*。记录中的域有时又叫做数据库对象的*属性*。

例如，考虑图12-7所示的数据库表，其中包含的是有关电影的信息。表中的每一行对应一条记录。表中的每个记录由相同的域构成，其中存储了特定的值。也就是说，每条电影记录都包括`MovieId`域、`Title`域、`Genre`域和`Rating`域，存放了这条记录特有的数据。数据库表都有一个名字，在这个例子中是`Movie`。

通常，表中会有一个或多个域被标识为**键**域。键域在表的所有记录中唯一标识了这个记录。也就是说，存储在表的每条记录的键域中的值必须是唯一的。在`Movie`表中，`MovieId`域是键的合理选择，因为两部电影可能重名，当然`Genre`和`Rating`域在这个例子中也不适合作为键域。

> **键**（key）：在表的所有记录中唯一标识一个数据库记录的一个或多个域。

键域`MovieId`中的每个值都必须是唯一的。大多数DBMS可以自动生成这种域，以确保实体的唯一性。不过这并不要求键值是连续的。上表中的最后三个实体包含的是截然不同的电影标识编号。只要它们是唯一的，`MovieId`域就可以作为键。

Movie

MovieId	Title	Genre	Rating
101	Sixth Sense, The	thriller horror	PG-13
102	Back to the Future	comedy adventure	PG
103	Monsters, Inc.	animation comedy	G
104	Field of Dreams	fantasy drama	PG
105	Alien	sci-fi horror	R
106	Unbreakable	thriller	PG-13
107	X-Men	action sci-fi	PG-13
5022	Elizabeth	drama period	R
5793	Independence Day	action sci-fi	PG-13
7442	Platoon	action drama war	R

图12-7 由记录和域构成的数据库表

图12-7中的电影表碰巧是按照`MovieId`值的升序排列的，也可以用其他方式（如电影名的字母顺序）排列表中的记录。在这个例子中，数据表中的行之间没有任何内在的关系。关系数据库表只是数据的逻辑视图，与底层的物理组织（记录是如何存储在硬盘上的）毫无关系。只有在查询数据库时，记录的排序才比较重要，例如查询所有`Rating`是PG的电影，这时我们可能想按照电影名对查询的结果排序。

表的结构反映了它所表示的模式。也就是说，模式是表中的记录的属性的表达式。可以如下表示上例中的数据库的模式：

```
Movie (MovieId:key, Title, Genre, Rating)
```

有时，在模式表示法中还会说明每个域存储的数据的类型，如数字或文本，此外还可能说明某个域可用的值集合。例如，在这个例子的模式中，可以说明`Rating`域的值只能是G、PG、PG-13、R或NC-17。整个数据库的模式由其中每个表的模式构成。

假设我们想创建一项电影租赁业务。除了出租的电影的清单外，还要创建一个客户信息表。图12-8中的表`Customer`存放了客户的信息。

Customer

CustomerId	Name	Address	CreditCardNumber
101	Dennis Cook	123 Main Street	2736 2371 2344 0382
102	Doug Nickle	456 Second Ave	7362 7486 5957 3638
103	Randy Wolf	789 Elm Street	4253 4773 6252 4436
104	Amy Stevens	321 Yellow Brick Road	9876 5432 1234 5678
105	Robert Person	654 Lois Lane	1122 3344 5566 7788
106	David Coggin	987 Broadway	8473 9687 4847 3784
107	Susan Klaton	345 Easy Street	2435 4332 1567 3232

图12-8 存放客户数据的数据库表

与Movie表一样，Customer表也有一个键CustomerId域。某些CustomerId的值与MovieId的值相同，这无关紧要。键的值只需要在同一个表中是唯一的。

在真实的数据库中，最好把Name域分为FirstName和LastName两个域。此外，完整的地址也可能会被分为几个部分，如City和State。在这个例子中，我们尽量把事情简单化。

Movie表和Customer表说明了如何用独立的表中的记录组织数据。不过，关系数据库管理系统的强大之处在于创建能把各个表从概念上联系起来的表，下一节将讨论这项功能。

12.3.2 关系

回顾一下，记录表示的是独立的数据库对象，记录的域是这些对象的属性。可以创建一个记录来表示对象之间的关系，包括记录中的属性之间的关系。因此，可以用一个表来表示对象间的关系的集合。

继续使用上面关于出租电影的例子，我们要能够表示特定的客户租了哪些电影。由于“租用”是客户和电影之间的关系，所以可以把它表示为一个记录。租用的日期和到期日是这种关系的属性。图12-9中的表Rents就是表示当前被租用的电影的关系记录的集合。

Rents

CustomerId	MovieId	DateRented	DateDue
103	104	3-12-2006	3-13-2006
103	5022	3-12-2006	3-13-2006
105	107	3-12-2006	3-15-2006

图12-9 存储当前被租用的电影的数据库表

Rents表包含有关关系中的对象（客户和电影）的信息和关系的属性。不过要注意，它并非包含客户和电影的所有数据。在关系数据库中要尽量避免数据重复。例如，在Rents表中没有必要保存客户的名字和地址。Customer表已经存储了这些数据。当需要这些数据时，用存储在Rents表中的CustomerId来检索Customer表，查找该客户的详细信息即可。同样地，当需要有关电影的信息时，用MovieId检索Movie表即可。

注意，CustomerId的值中出现了两次103。这说明同一个客户可以租借两部电影。

数据库表中的数据会根据需要被修改、添加和删除。当给库存添加了电影或从中删除电影时，Movie表中的记录都要更新。当有新客户加入时，需要把他们添加到Customer表中。随着电影不断地被租出去或还回来，还要添加或删除Rents表中的记录。

通用产品代码

在查看产品包装时，你会发现通用产品代码（UPC）和与之关联的条形码，如插图所示。使用UPC代码是为了加速在商店购买商品的速度，以及帮助跟踪库存。

UPC符号由机器能识别的条形码和人能识别的12位UPC编号组成。UPC编号的前6位数字是制造商的身份编号。例如，General Mills的制造商身份编号是016000。接下来的5位数字是项目编号。每种类型的产品和同一产品的不同包装

都有一个唯一的项目编号。因此，两升装的可口可乐与两升装的减肥可乐项目编号不同，10盎司装的Heinz番茄酱与14盎司装的项目编号也不同。

最后一位UPC代码叫做校验数位，扫描器可以用它确定扫描的UPC编号是否正确。校验数位是通过对UPC编号的其余数位的计算得来的。在读入UPC编号后，扫描器将对它执行运算，用校验数位进行验证（更多有关校验数位的信息请参阅第17章）。

针对某些产品，尤其是小产品，开发了新的UPC编号法，通过减少某些数位从而缩短了UPC编号。采用这种方法，可以减小整个UPC符号的大小。

注意，UPC编号并不存储产品的价格。当收银机（更正式的叫法是电子收款系统，POS）扫描一个产品时，它将使用UPC编号中的制造商编号和项目编号在数据库中检索该项目，而数据库则包含大量的产品信息，包括它的价格。UPC中只保留基本的信息，这样不必更改产品的标签，就可以很容易地更改其他信息（如价格）。不过，无论是否是有意的，这样容易产生“扫描器欺骗”的问题，即一个产品在数据库中的价格与它在货架上的价格不符。

12.3.3　结构化查询语言

结构化查询语言（SQL）是一种用于管理关系数据库的综合性数据库语言。它包括指定数据库模式的语句和添加、修改及删除数据库内容的语句。顾名思义，它还具有查询数据库以获取特定数据的功能。

> **结构化查询语言**（Structured Query Language, SQL）：用于管理和查询数据的综合性关系数据库语言。

SQL的原始版本是20世纪70年代IBM开发的Sequal语言。1986年，美国国家标准化组织（ANSI）发布了SQL标准，这是访问关系数据库的商用数据库语言的基础。

SQL不区分大小写，因此其中的关键字、表名和属性名可以是大写的、小写的或大小写混合的。空格被用作语句中的分隔符。

查询

首先我们来介绍简单的查询。select语句是查询的主要工具。基本的select语句包括一个select从句、一个from从句和一个where从句：

select *attribute-list* from *table-list* where *condition*

select从句决定了返回哪些属性。from从句决定了使用哪个表进行查询。where从句限制了返回的数据。例如：

```
select Title from Movie where Rating = 'PG'
```

这个查询的结果是Movie表中所有Rating为PG的电影名的列表。如果不需要特殊的限制，可以省略where从句：

```
select Name, Address from Customer
```

这个查询返回的是Customer表中所有客户的名字和地址。select从句中的星号（*）表示要返回选中的记录中的所有属性：

```
select * from Movie where Genre like '%action%'
```

这个查询返回的是Movie表中Genre属性包含单词action的记录的所有属性。SQL中的like操作执行的是字符串的模式匹配，符号%与任何字符串都匹配。

select语句还可以用order从句指定查询结果的排序方法：

```
select * from Movie where Rating = 'R' order by Title
```

这个查询返回的是Rating为R的电影的所有属性，按照电影名的字母顺序排列。

SQL支持的select语句的变体比我们这里介绍的多得多。我们的目标只是给你介绍数据库的概念。要精通SQL，你还需要掌握更多的细节。

修改数据库的内容

用SQL中的insert、update和delete语句，可以改变表中的数据。insert语句可以给表添加一条新记录。每个insert语句都指定了新记录的属性值。例如：

```
insert into Customer values (9876, 'John Smith'
'602 Greenbriar Court', '2938 3212 3402 0299')
```

这个语句将在Customer表中插入一条指定了属性的新记录。

update语句可以改变表中的一条或多条记录的值。例如：

```
update Movie set Genre = 'thriller drama' where title = 'Unbreakable'
```

这个语句将把电影Unbreakable的Genre属性改为thriller drama。

delete语句可以删除表中与指定的条件匹配的所有记录。例如，如果要删除Movie表中所有Rating为R的电影，可以使用下列delete语句：

```
delete from Movie where Rating = 'R'
```

与select语句一样，insert、update和delete语句也有许多变体。

SQL的数学基础

SQL的操作中混有代数，用于访问和操作关系表中的数据。这种代数是20世纪60年代末期由IBM的E.F.Codd定义的，他的贡献使他赢得了1981年的图灵奖。SQL的基本操作包括：

- Select操作，用于识别表中的记录。
- Project操作，用于生成表列的子集。
- 笛卡儿乘积操作，用于连接两个表的行。

其他还有集合操作联合、求差、求交集、自然连接（笛卡儿乘积的子集）和除法操作。

12.3.4 数据库设计

要想使数据库完成自己的任务，那么从开始就要认真设计它。早期拙劣的计划会导致数据库不能支持必要的关系。

一种常用的设计数据库的方法叫做**实体关系（ER）建模法**。ER建模的主要工具是**ER图**。ER图用图形化的形式捕捉了重要的记录类型、属性和关系。数据库管理员可以根据ER图定义必要的模式，创建适合的表来支持由图指定的数据库。

实体关系（ER）建模法（entity-relationship (ER) modeling）：设计关系数据库的常用方法。

ER图（ER diagram）：ER模型的图形化表示法。

图12-10所示的ER图展示了租赁电影这个例子的各个方面。ER图用特定的形状来区分数据库的不同部分。矩形表示记录的类型（可以把它看作数据库对象的类）。椭圆表示记录的域（或属性）。菱形表示关系。

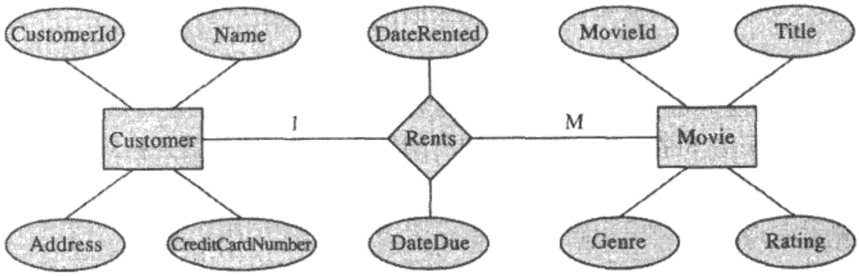


图12-10　电影租赁数据库的ER图

虽然ER图中各个元素的位置并不重要，但认真布置它们，会更易于阅读。注意，像`Rents`这样的关系也可以有自己的属性。

此外还要注意关系连接线上的标签，一边是1，另一边是M。这些标号说明了关系的**基数约束**。基数约束限制了一次可以存在的关系数量。

> **基数约束**（cardinality constraint）：在ER图中，一次可以存在于两个实体间的关系数量。

一般的基数关系有三种：

- 一对一
- 一对多
- 多对多

客户和电影之间的关系是一对多的。也就是说，一位客户可以租借多部电影，但（任何时刻）一部电影却只能借给一位客户。基数约束有助于数据库设计者表达关系的细节。

12.4　信息安全

管理信息的系统都必须解决对信息的某个级别的安全保护问题。信息安全是由组织或个人强加的一组技术或策略，以确保对受保护的数据的正确访问。它能够确保未经授权的用户不能读取或修改数据，数据只在需要的时候由有需要的人访问。要很好地实现这点是很有挑战性的。

12.4.1　机密性、完整性和可用性

信息安全可以描述为机密性、完整性和可用性的综合体，所以叫做CIA三元组，如图12-11所示。虽然信息安全的这些方面部分相同而且互相影响，但它们还是定义了三种特定的方式来看问题。任何好的信息安全问题的解决方案都必须有效地解决这三方面的问题。

机密性可以确保关键数据免受未授权的访问。例如，你一定不想让你的银行账户余额被别人知道。

> **信息安全**（information security）：确保数据的正确访问的技术和策略。
> **机密性**（confidentiality）：确保数据免受未授权的访问。

完整性确保了只有正确的访问机制才能修改数据。它可以在信息中定义信任级别。当然，你一定不想让黑客修改你的存款额，也一定不想让银行出纳员（授权用户）在未经你允许的情况下错误地修改你的存款额。更进一步来说，你也不想在数据的电子传输过程中由于掉电而导致存款额改变。

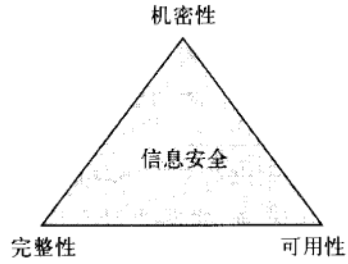


图12-11 信息安全的CIA三元组

可用性是指授权的用户能够在必要时以合法目的访问适当的信息的程度。即使数据已经受保护了，但是如果不能得到它，它也是无用的。如果预防措施没有备份数据并且没有维护冗余访问机制，那么硬件问题（如硬盘损坏）就会引发可用性问题。此外，黑客还可能发动拒绝服务（DoS）攻击，让无用的传输占据网络，从而使合法用户无法连接到远程系统。

从商业角度来看，信息安全规划需要进行**风险分析**，即确定哪些数据需要保护、标识数据的风险以及计算风险变为现实的可能性的过程。一旦完成了风险分析，就可以实施信息安全计划以相应地管理风险。大多数风险带给我们的威胁是由系统的弱点造成的，我们想最小化这些弱点。这些威胁有恶意的（如黑客攻击），也有意外的（如系统崩溃）。

> **完整性**（integrity）：确保只有正确的访问机制才能修改数据。
> **可用性**（availability）：授权用户能够为合法目的而访问数据的程度。
> **风险分析**（risk analysis）：判断关键数据的特性和发生风险的可能性。

信息安全专家信奉的另一条原则是把数据的管理权限分离开，这样可以确保单一用户不会对系统造成重大影响。通常是通过给关键操作设置冗余的检查和/或审批权限来实现这一原则。例如，大型的金融交易通常需要一个单独的审批流程。管理员应该只给每个用户分配执行其工作所必需的权限。

在计算机的各个层面都存在一定的技术解决方案来支持信息安全。在第10章和第11章中，我们讨论过关于操作系统的安全问题，如防止程序和用户访问内存中其不该访问的区域。在第14章中，我们将从整体上讨论计算机的安全性，包括计算机系统必须防御的常见攻击类型。第15章则会讨论与网络相关的安全性，如防火墙的使用。

接下来，让我们看一种与信息安全息息相关的技术——密码学。

12.4.2 密码学

密码学是与信息加密相关的学术研究领域。单词“cryptography”源自希腊语中的“secret writing”一词。密码学的基本概念已经以多种形式被应用了几千年，以帮助人们保密，避免信息落入错误的人手里。

所谓**加密**，是把普通的文本（用密码学术语来讲即明文）转换成难以理解的密文形式的过程。所谓**解密**，是加密过程的逆过程，即把密文转换成明文。**密码**是用于执行特定的加密/解密过程的算法。密码的**密钥**是指导该算法的一组特定参数。

你可能已经尝试过各种密码。顾名思义，**替换密码**就是把明文中的一个字符替换成另一个字符。接收者执行反向的替换就能破解消息。

最著名的替换算法可能非**凯撒密码**莫属，它是Julius Caesar用于和他的将领们通信的一种

密码。Caesar只是把消息中的字符在字母表中移动特定的位数。例如，把字符向右移动5位，就会产生下面的替换算法：

原始值：A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

替换值：F G H I J K L M N O P Q R S T U V W X Y Z A B C D E

> **密码学**（cryptography）：与信息加密相关的学术研究领域。
>
> **加密**（encryption）：把明文转换成密文的过程。
>
> **解密**（decryption）：把密文转换成明文的过程。
>
> **密码**（cipher）：用于加密和解密文本的算法。
>
> **密钥**（key）：指导密码的一组参数。
>
> **替换密码**（substitution cipher）：把一个字符替换成另一个字符的密码。
>
> **凯撒密码**（caesar cipher）：把字符在字母表中移动特定位数的替换密码。

用这种方法，消息“MEET ME AT THE OLD BARN”就会被加密成

```
RJJY RJ FY YMJ TQI GFWS
```

这种密码的密钥由移动的位数和方向（向左或向右）构成。当然，空格可以不在加密范围内，或者被替换成其他字符，也可以加入标点。还有很多其他的替换密码，如把一组字母替换成一个单元或者在消息的不同点执行不同的替换方法。

错乱密码是以某种方式重新排列消息中的字符。例如，**路线密码**是把消息排列成一个字符阵列，然后指定阵列中的一种路线来加密消息的错乱密码。要加密消息“MEET ME AT THE OLD BARN”，可以把字母排成阵列

```
M T A H L A
E M T E D R
E E T O B N
```

可以按从右上角顺时针内螺旋的路线进行加密，生成的消息如下：

```
ARNBOTEEEMTAHLDETM
```

接收到该消息后，会重构这个阵列，从下向上读字母。这个密码的密钥由阵列的维数和用于加密的路线构成。在构建阵列时，如果字符的数量不能刚好构建出特定维数的阵列，可以用额外的字符作为占位符。

所谓**密码分析**，是“破解”用密码编写的代码的过程。简而言之，就是在不知道密码或密钥的情况下尝试推断出明文。诸如替换密码和错乱密码这样的老密码术对于现代计算机来说毫无挑战性。已经有程序可以相当容易地判断出采用的是哪种类型的加密方法，并且生成对应的明文消息。针对现代计算学，需要更复杂的加密方法。

> **错乱密码**（transposition cipher）：重新排列消息中的字符顺序的密码。
>
> **路线密码**（route cipher）：错乱密码的一种，把消息排成阵列，用特殊的路线遍历它。
>
> **密码分析**（cryptanalysis）：在不知道加密采用的密码或密钥的情况下解密消息的过程。

这些方法的另一个缺点是发送者和接收者要共享密钥，而密钥还要保密。这个共享密钥就成了整个过程的弱点，因为它必须告诉接收双方，这就可能会被中途截取。如果密钥泄露

了，那么未来所有被加密的消息就都会有危险。

让我们来看一个现代加密学的方法，它将这种弱点的风险降至最低。在**公共密钥加密法**中，每个用户有一对密钥，这对密钥有一定的数学相关性。这种关系非常复杂，用一个密钥加密的消息只能用与这个密钥配对的密钥解密。其中一个密钥是*公共密钥*，可以随意发布，另一个密钥则是*私有的*。

假设两个用户（Alice和Bob）想安全地通信。记住，他们有自己的公共密钥和私有密钥对。要给Bob发送消息，Alice首先要获得Bob的公共密钥来加密消息，该密钥可以轻易获得。现在，除了Bob之外，没有人能够解密这条消息了，甚至是Alice。然后Alice把消息安全地发送给Bob，Bob用自己的私有密钥解密这条消息。

同样，Bob只有在用Alice的公共密钥加密消息后才把它发送给Alice。Alice用自己的私有密钥解密消息。只要Alice和Bob都保守自己的私有密钥，那么谁有公共密钥都无关紧要了。

公共密钥加密法还带动了**数字签名技术**的使用，这种技术通过在消息后附加数据来“签署”文档，这种签名不仅对发送者来说是唯一的，而且还非常难伪造。有了数字签名，接收者就能验证消息是否真的来自署名的发件人，是否在传输过程中被第三方修改了。这种签名是使用软件创建的，该软件把消息压缩到一种叫做*消息摘要*的表格中，然后用发送者的私有密钥加密消息摘要。接收者用发送者的公共密钥解密消息摘要，然后用它与使用消息自身创建的摘要进行比较。如果两者吻合，那么该消息可能就是真实的、未被改变的。

公共密钥加密法的核心是公共密钥能够公开，可以随意发布。那么如果某人以别人的名义创建了一个密钥对会怎么样呢？接收者如何判断一个公共密钥是否是可信的？一些组织通过建立认证授权中心来处理这类风险，这些中心会为可信的发送者创建**数字认证**。这个认证是由发送者的个人数据和经过鉴定的公共密钥构成的。然后，当一个新消息到达时，就会使用数字认证对它进行验证。如果你不具有消息发送者的数字认证，那么就要决定是否信任这条消息。

> **公共密钥加密法**（public-key cryptography）：密码学的一种方法，每个用户具有两个相关的密钥：一个是公开的，另一个是私有的。
>
> **数字签名**（digital signature）：附加在消息上的数据，由消息自身和发送者的私有密钥生成，以确保消息的真实性。
>
> **数字认证**（digital certificate）：发送者经过鉴定的公共密钥的一种形式，用于最少化恶意伪造事件。

小结

信息系统是让用户组织和管理数据的应用软件。一般信息系统软件包括电子制表软件和数据库管理系统。其他领域（如人工智能）也有自己专用的数据管理技术。

电子制表软件是用单元格来组织数据和用于计算新值的公式的应用软件。用行列标号可以引用单元格，如A5或B7。单元格可以存放基本数据或公式。公式通常会引用其他单元格中的值，还会使用内置函数来计算结果。此外，公式还可以使用一个单元格范围内的数据。如果单元格中存放的是公式，那么真正显示的是公式计算出的值。对于电子数据表中的公式，避免循环引用（两个或多个单元格的计算结果要互相依赖）很重要。

电子数据表具有多功能性和可扩展性。它们适用于多种不同的情况，能够对变化动态地作出响应。如果电子数据表中的值被改变了，相关的公式会自动重新计算，生成最新的结果。如果给电子数据表添加了行或列，那么公式的范围也会被立刻校正。电子数据表尤其适用于模拟假设分析，其中的假设值将被不断修改，以了解对系统其他数据的影响。

数据库管理系统包括存储数据的物理文件、支持数据访问和修改的软件以及指定数据库的逻辑布局的数据库模式。关系模型是目前最常用的数据库模型。它用表组织数据，表由记录（对象）构成，记录由域（属性）构成。每个表会被指派一个（或一组）键域，键的值唯一标识了表中的每个记录。

数据库元素之间的关系可以用新的表表示，这个表也可以有自己的属性。关系表并不是重复其他表的数据，而是存储数据库记录的关键值，以便需要的时候能够查找详细的数据。

结构化查询语言（SQL）是查询和操作关系数据库的标准语言。select语句用于查询操作，它具有很多变体，能够访问数据库中的特定数据。其他SQL语句能对数据库执行添加、修改和删除操作。

数据库一定要仔细设计。实体-关系建模法和ER图是常用的数据库设计方法。ER图图形化地描述了数据库对象之间的关系，说明了它们的属性和基数约束。

数据库中的信息安全已经成为一个主要问题。输入的数据正确，只有具有访问权限的用户才能访问信息，未授权的访问是不允许的，这些都非常重要。

道德问题：加密

你曾经在Internet上买过东西吗？使用过网上银行吗？或者通过网站传输过医疗记录吗？你对这些事务的安全性有多少信心呢？随着网络不断全球化，把敏感信息从一台计算机安全地传递到另一台计算机的能力显得至关重要。例如，在线购物时你提供的个人信息。电子商务网站常常要求提供信用卡号、地址、电话、电子邮件地址和其他一些销售所需的信息，如年龄、性别、收入或者兴趣爱好等。那么谁能访问这些数据呢？

安全的网站通过密码学来保护个人信息。密码学的基本思想源自罗马时代，Caesar用简单的字母编码对自己的通信进行加密。今天，加密（encryption）作为密码学的一种方法，被用于对通过Internet发送的消息进行编码。一旦消息加密了，就只有用密钥才能解密。加密的目的是加强网络安全性，使得除了指定的接收者外，没有人能够访问传输的资料。

强大的加密技术可以提高在线客户的信心，不过许多人也担心复杂的加密技术能帮助罪犯、黑客、间谍和恐怖主义者。例如，美国政府对加密技术的出口进行了某些限制，有些官员还游说议员颁布法令，限制加密技术的产品在美国的使用强度。20世纪90年代，FBI支持一项政策，如果提出请求，公民就要交出解密密钥。政府还能通过“后门”访问受保护的信息，这样无需解密密钥就可以访问受保护的数据。

隐私权支持者反对这样的加密限制措施。他们认为政府监控加密技术的企图本质上是严格统治。此外，他们还认为后门为黑客打开了安全站点，强大的加密技术可以防止罪犯接触机密信息。

2001年9月11日恐怖分子对美国的袭击使这场关于加密技术的争论进入了白热化。毫无疑问，通信是同步和执行恐怖袭击的关键。恐怖组织的成员可能使用了加密技术或者把消息隐藏在图像中的方法进行电子通信，而美国政府却没能截获和解密这些信息。当然，现在推测访问加密密钥和提高解密能力是否能改变9 · 11事件已经于事无补，重要的是探讨政府应该对于加密的通信具有哪种类型的控制。

练习

判断练习1～23中的陈述的对错：

A. 对　　B. 错

1. 电子数据表中的单元格只能存放原始数据。
2. 可以以各种方式格式化电子数据表中的值。
3. 应该使电子数据表能够自动反映出数据的变化。
4. 电子数据表函数是用户为计算而编写的程序。
5. 可以指定一行单元格作为单元格的范围，也可以指定一列，但是不能同时用行列指定一块单元格。
6. 电子数据表中的循环引用是很强大、很有用的特性。
7. 电子数据表对执行模拟假设分析很有用。
8. 模拟假设分析一次只会影响电子数据表中的一个值。
9. 数据库引擎是支持对数据库内容的访问的软件。
10. 物理数据库表示了数据库中数据的逻辑结构。
11. 查询是对数据库信息的请求。
12. 可以采用多种方式结构化查询的结果。
13. 分级模型是目前最常用的数据库管理模型。
14. 数据库表是记录的集合，记录是域的集合。
15. 表的键域的值能唯一标识表中的一个记录。
16. 数据库引擎要访问和修改数据库，通常需要与一种特定的语言交互。
17. 实体-关系（ER）图以图形的形式表示了主要的数据库元素。
18. 关系的基数限制了一次能够存在的关系数量。
19. 信息完整性确保了只有正确的机制才能修改数据。
20. 把威胁与弱点配对是风险分析的一部分。
21. 解密是把明文转换成密码的过程。
22. 错乱密码是现代加密学的一个实例，计算机很难破解它。
23. 有了数字签名，接收者就能验证消息是否真的来自于所署的发件人。

为练习24～28中的问题找到匹配的答案。

A. 动态的　　B. 函数

C. 循环　　D. 范围

E. 模式　　F. 域

24. 电子数据表是______，因为它能够立即响应数据的变化，更新所有受影响的数据。
25. 电子数据表的公式可以对单元格的______进行操作，如C4..C18。
26. 数据库______是数据库中的数据的逻辑结构规约。
27. 当一个公式的结果最终由另一个公式决定，反之亦然时，将发生______引用。
28. ______只包含一个数据值。

练习29～73要使用下列学生成绩单的电子数据表。

	A	B	C	D	E	F	G	H
1					Grades			
2				*Exam 1*	*Exam 2*	*Exam 3*	*Average*	
3								
4			Bill	89	33	80	67.3333	
5			Bob	90	50	75	71.6666	
6			Chris	66	60	70	65.3333	
7			Jim	50	75	77	67.3333	
8		Students	Judy	80	80	80	80	
9			June	83	84	85	84	
10			Mari	87	89	90	88.6666	
11			Mary	99	98	90	95.6666	
12			Phil	89	90	85	88	
13			Sarah	75	90	85	83.3333	
14			Suzy	86	90	95	90	
15		Total		893	839	912	881.333	
16		Average		81.1818	76.2727	82.9090	80.1212	

29. Exam 2的成绩是多少？
30. Exam 1的平均分是多少？
31. Sarah的平均分是多少？
32. Mari第三次测验的成绩是多少？
33. Suzy的测验成绩是多少？
34. `F15`存放的公式是什么？
35. `D16`存放的公式是`D15/COUNT(D4..D14)`，与之计算结果相同的公式是什么？
36. `E13`存放的公式是什么？
37. 如果Phil的Exam 2成绩被更正为87，那么哪些值会改变？
38. 什么是电子数据表循环引用？它有什么问题？
39. 图12-5展示了一个非直接的循环引用，再给出一个这样的例子。
40. 什么是模拟假设分析？
41. 如果要用电子数据表来制定计划，跟踪某些股票的购买情况，请列举一些模拟假设分析的问题。请解释如何建立一个电子数据表来帮助回答这些问题。

练习42～45要使用本书的站点上提供的数据表表单或使用真正的电子制表软件来设计电子数据表。对于这些问题，你的导师可能会提供更明确的指示。

42. 设计一个电子数据表，记录你喜欢的棒球联赛队的统计信息，包括跑垒、击球、失误和击球跑垒得分的数据。计算每个队员的统计数据和整个队的统计数据。
43. 设计一个电子数据表，维护一组学生的成绩表。包括测验和项目的成绩，在计算每位学生的最终成绩时要进行加权。计算全班学生每次测验和每个项目的平均分。
44. 假设你要进行一次商务旅行。设计一个电子数据表记录你的开销，创建一个总计值。要包含旅行的方方面面，如车程、机票费用、酒店费用等（如出租车费和小费）。
45. 设计一个电子数据表，预测一个特定项目的活动，然后跟踪这些活动。列出这些活动、预测的活动日期和实际的活动日期以及时间安排的偏差。还可以添加其他合适的数据。
46. 比较数据库和数据库管理系统。
47. 什么是数据库模式？
48. 描述关系数据库的一般组织形式。
49. 什么是数据库的域（属性）？
50. 在图12-7的数据库表中还可以再加入哪些域（属性）？
51. 在图12-8的数据库表中还可以再加入哪些域（属性）？
52. 什么是关系数据库表的键？
53. 请说明图12-8中的数据库表的模式。
54. 在关系数据库中如何表示关系？
55. 定义一个SQL查询，返回`Customer`表中的所有记录的所有属性。
56. 定义一个SQL查询，返回`Rating`为R的所有电影的id和名字。
57. 定义一个SQL查询，返回`Customer`表中住在Lois Lane的每位客户的地址。
58. 定义一个SQL语句，把电影`Armageddon`插入`Movie`表。
59. 定义一个SQL语句，更改`Customer`表中Amy Stevens的地址。
60. 定义一个SQL语句，删除客户id是103的客户。
61. 什么是ER图？
62. 在ER图中如何表示实体和关系？
63. 在ER图中如何表示属性？
64. 什么是基数约束？在ER图中如何表示它们？
65. 三种一般的基数约束是什么？
66. 设计一个数据库，存储有关图书馆中的图书的数据、使用它们的学生的数据和借书的数据。创建这个数据库的ER图和样表。
67. 设计一个数据库，存储有关大学开设的课程的数据、教授这些课程的老师的数据和选择这些课程的学生的数据。创建这个数据库的ER图和样表。
68. 信息安全的CIA三元组是什么？
69. 除了本章中给出的实例，再列出三个数据完整性冲突的例子。
70. 使用凯撒密码，把三个字母右移，加密消息“WE ESCAPE TONIGHT”。
71. 使用本章中描述的凯撒密码，解密消息“WJNSKTWHJRJSYX FWWNAJ RTSIFD”。
72. 使用本章中用到的错乱加密技术，加密消息“WHO IS THE TRAITOR”。
73. 描述Claire如何用公共密钥加密技术向David发送消息。

思考题

1. 除了本章列举的例子，另外想出5个需要使用电子数据表的情况。
2. 除了本章列举的例子，另外想出5个需要建立数据库的情况。
3. 使用计算机化的数据库是不是就意味着可以抛弃文件夹呢？哪种类型的文件夹仍然是必需的？
4. 什么是密码学？作为一个学生，它与你有什么关系？
5. 密码学与电子商务有什么关系？
6. 密码学与国防有什么关系？
7. 当前关于密码学的争论是什么状况？9 · 11事件对这一争论有什么影响？

第13章 人工智能

计算的一个子学科——人工智能（artificial intelligence, AI）在许多方面都非常重要。它向许多人展示了计算的未来，计算机发展得更像人类了。对另一些人来说，人工智能则是应用新技术来解决问题的途径。

提到人工智能，可能会唤起你各种各样的联想，如会下棋的计算机或者能够做家务的机器人。这些当然属于人工智能，不过人工智能还远远不止于此。从普通的到怪异的，AI对许多类型的应用程序的开发都有影响。人工智能打开了新世界的大门，这是计算领域的其他子学科做不到的。它在采用最新技术的应用程序开发中扮演着至关重要的角色。

目标

学完本章之后，你应该能够：

- 区分人类可以解决得最好的问题和计算机能够解决得最好的问题。
- 解释图灵测试。
- 定义知识表示的意义，并说明在语义网中如何表示知识。
- 为简单的情况开发检索树。
- 解释专家系统的处理。
- 解释生物神经网络和人工神经网络的处理。
- 列出自然语言处理的各个方面。
- 解释自然语言理解中的各种二义性。

13.1 思维机

计算机是令人吃惊的设备。它们可以绘制复杂的三维图像，处理整个公司的工资表，判断正在建造的大桥是否能承受预计的交通压力。然而要它们理解一个简单的对话却很困难，它们可能分不清什么是桌子，什么是椅子。

当然，有些事计算机会比人类做得好。例如，要用纸和笔求1000个4位数的加法，虽然你也可以做，但是要花很长的时间，还很可能出错。计算机却只要不到1秒的时间就能给出准确无误的计算结果。

但是，如果要指出图13-1中所示的猫，你会毫不犹豫地指出它。而计算机就很难做到这一点，而且很可能出错。人类对这种类型的问题具有大量的知识和推理能力，我们仍然在努力尝试用计算机执行类似人类的推理。

在现代技术中，虽然计算机擅长计算，但却不擅长需要智能的任务。**人工智能**就是研究对人类思想建模和应用人类智能的计算机系统的学科。

> **人工智能**（artificial intelligence, AI）：研究对人类思想建模和应用人类智能的计算机系统的学科。



图13-1 计算机要识别图片中的猫会比较困难

13.1.1 图灵测试

1950年，英国数学家Alan Turing发表了一篇具有里程碑性质的论文，其中提出了一个问题：机器能够思考吗？在慎重地定义了术语智能和思维之后，最终他得出的结论是我们能够创造出可以思考的计算机。但他又提出了另一个问题：如何才能知道何时是成功了呢？

他对这个问题的答案叫做**图灵测试**，根据经验来判断一台计算机是否达到了智能化。这种测试的基础是一台计算机是否能使人们相信它是另一个人。

> **图灵测试**（turing test）：一种行为方法，用于判断一个计算机系统是否是智能的。

虽然多年来出现了各种图灵测试的变体，但这里的重点是它的基本概念。图灵测试是这样建立的，由一位质问者坐在一个房间中，用计算机终端与另外两个回答者A和B通信。质问者知道一位回答者是人，另一位回答者是计算机，但是不知道究竟哪个是人，哪个是计算机。如图13-2所示。

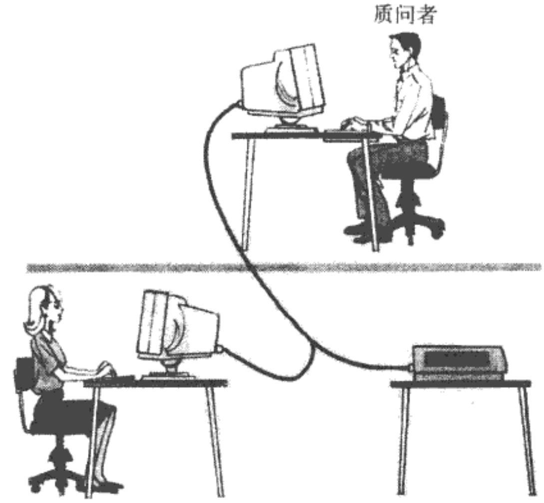


图13-2 在图灵测试中，质问者必须判断出哪个回答者是计算机，哪个是人

分别与A和B交谈之后，质问者要判断出哪个回答者是计算机。这一过程将由多个人反复执行。这个测试的假设是如果计算机能瞒过足够多人，那么就可以把它看作是智能的。

有些人认为图灵测试很适合测试计算机的智能，因为它要求计算机处理各种各样的知识，还要具有处理交谈中的变化所必需的灵活性。要瞒过质问人，计算机需要掌握的不仅仅是事实知识，还要注意人的行为和情绪。

另一些人则认为图灵测试并不能说明计算

机理解了交谈的语言，而这一点对真正的智能来说是必需的。他们提出，程序能够模拟语言的内涵，可能足够使计算机通过图灵测试，但只凭这一点并不能说计算机智能化了。

通过图灵测试的计算机具有**弱等价性**，即两个系统（人和计算机）在结果（输出）上是等价的，但实现这种结果的方式不同。**强等价性**说明两个系统使用的是相同的内部过程来生成结果。有些AI研究员断言，只有实现了强等价性（即创造出了能像人一样处理信息的机器），才可能存在真正的人工智能。

> **弱等价性**（weak equivalence）：两个系统基于结果的等价性。
>
> **强等价性**（strong equivalence）：两个系统基于结果和实现这种结果的处理方法的等价性。

纽约的慈善家Hugh Loebner组织了首次正式的图灵测试。从1991年起，每年举行一次这样的竞赛，其中反应与人类的反应最难区别的计算机将获得100 000美元的奖金和一块金牌。迄今为止，奖金争夺战仍在进行中。此外，每年还会给予相对来说最像人类的计算机2000美元的奖金和一块铜牌。对于热衷人工智能的人来说，Loebner奖比赛已经成了每年重要的赛事。

目前已经开发了各种程序来执行这种人机交互，它们通常叫做**聊天机器人**。从万维网上可以找到许多这样的程序，它们都着重于某个特定的主题。如果这些程序设计得足够好，它们就可以执行合理的对话。不过，大多数情况下，用户用不了多久就能发现对话中的难用之处，这就暴露了人类的思维并不能决定反应的事实。

> **Loebner奖**（Loebner prize）：正式的图灵测试，每年举行一次。
>
> **聊天机器人**（chatbot）：用于执行人机对话的程序。

13.1.2 AI问题的各个方面

人工智能这个领域有许多分支。这一章的整体目标是让你了解人工智能涉及的主要问题以及还未解决的难题。本章余下的部分将探讨下列AI问题：

- 知识表示——用于表示知识，以便计算机能够用来解决智能问题的技术。
- 专家系统——嵌入人类专家知识的计算机系统。
- 神经网络——模拟人脑处理的计算机系统。
- 自然语言处理——处理人类用来交流的语言的难题。
- 机器人学——关于机器人的研究。

13.2 知识表示

表示一个对象或事件所需的知识会根据情况而有所不同。对于要解决的问题，我们需要特定的信息。例如，如果要分析家族关系，那么就要知道Fred是Cathy的父亲，至于Fred是水管工、Cathy有部掘土机这些信息就无关紧要了。而且我们需要的不仅仅是特定的信息，还需要一种形式，使我们能有效地检索和处理信息。

表示知识的方法有许多种。可以用自然语言描述知识。例如，可以用一段英文描述一个学生以及他与外界的联系。尽管自然语言的说明性很强，但它不容易处理。所以我们需要形式化的语言，这里用一个近似于数学符号的符号表示学生。这种形式化更适合严格的计算机

处理，但却难于理解和正确使用。

一般来说，我们想独立于数据的底层实现，创建它的逻辑视图，以便能用特定的方式处理数据。不过，在人工智能领域，我们想捕捉的信息常常会产生有趣的新数据表示法。我们想捕捉的不止是事实，还有它们之间的关系。要解决的问题的类型决定了要加于数据的结构。

当研究过特定的问题领域后，新的知识表示方法就会出现。这一节将分析其中的两种——语义网和检索树。

13.2.1 语义网

语义网是一种知识表示法，重点在于对象之间的关系。表示语义网的是有向图。图中的节点表示对象，节点之间的箭号表示关系。箭号上的标签说明了关系的类型。

> **语义网**（semantic network）：表示对象之间关系的知识表示法。

语义网借用了许多面向对象的概念（第6章和第8章介绍过），包括继承和实例化。继承关系说明一个对象是（is-a）另一个对象更具体的版本。实例化（instance-of）是一个真正的对象和这种对象的说明（如类）之间的关系。

图13-3展示了一个语义网，其中既有is-a关系，也有instance-of关系。此外，它还有其他类型的关系，如lives-in（John住在继承的产业中）等。在语义网中，关系的类型基本上没什么限制。

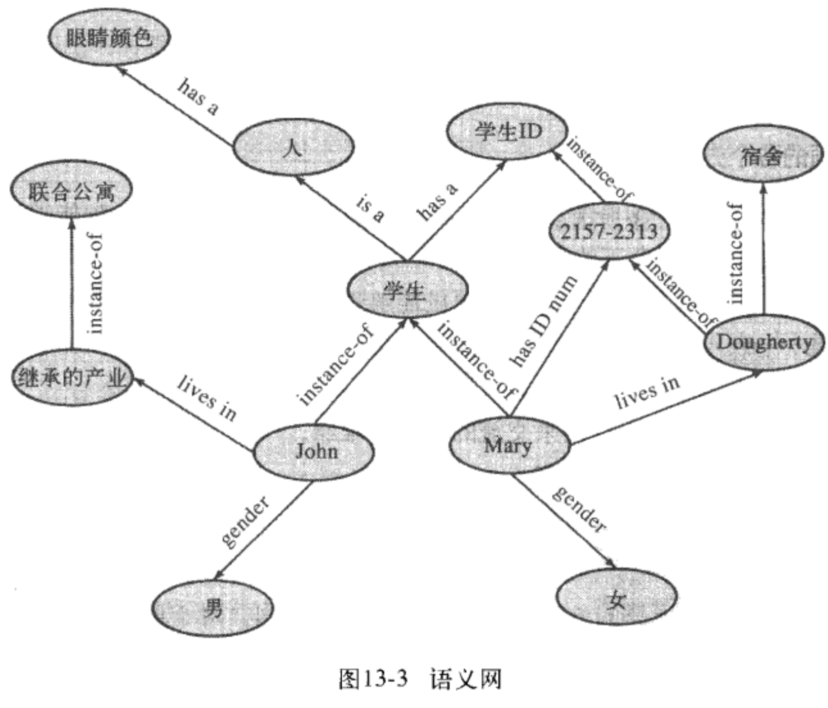


图13-3　语义网

在这个语义网中还可以表示更多的关系。例如，可以说明每个人是惯用左手还是惯用右手，或者说明John有一辆Honda牌的汽车，又或者说明每个学生的GPA。我们要表示的关系完全出于个人的选择，取决于回答我们面对的各种类型的问题所需要的信息。

建立关系的方法也有多种。例如，可以不说明每个学生所住的公寓，而说明每个公寓住

了哪些学生。换句话说，可以反转箭头，把lives-in关系改为houses关系。同样地，这种选择也是在设计语义网时由我们自己决定的。哪种方式更适合解决我们的问题呢？在某些情况下，我们会两者都选用。

Herbert A. Simon



Herbert A. Simon是我们这个时代一个多才多艺的人。他所获得的是政治学博士学位，但他的诺贝尔奖是有关经济学的，他的主页上包罗了计算机科学、心理学和哲学的内容。

Simon博士于1916年出生在密尔沃基。他的父亲是位工程师，最后成了一位职业律师，母亲是位有造诣的钢琴家。1936年，他收到了芝加哥大学颁发的学士学位，之后从事了几年编辑和行政工作。1943年，他在芝加哥大学攻读完了政治学的博士学位，从此开始了长达58年的学术生涯，其中最后52年都是在Carnegie Mellon度过的。

1955年，Simon博士和Allen Newell及J.C.Shaw（他们的程序员）一起编写了Logic Theorist，这个程序能够发现几何定理的证明。与此同时，Simon还致力于E. A. Feigenbaum on EPAM的开发，这个程序能够对人的感觉和记忆力建模。这些程序以及此后有关人类思想模拟、问题求解和言语学习的论文标志着人工智能领域研究的开端。1988年，ACM为Simon和Newell在人类问题求解方面所做的贡献授予他们图灵奖。1995年，在人工智能国际联合会议上，Simon获得了Research Excellence奖。

Simon博士在信息处理和决策方面的兴趣促使他建立了经济学理论“有限合理性”，为此他得到了1978年的诺贝尔经济学奖。经典的经济学认为人们会进行合理的选择，用最好的价格买到最好的物品。Simon博士的推理是“最好”的选择是不可能的，因为选择太多，根本没有时间对它们进行分析。因此，人们总是选择第一个足够满足他们要求的选项。他的诺贝尔获奖词是“为了他对经济组织内的决策过程所做的开拓性研究”。

在他漫长的职业生涯中，Simon博士一直保持着非凡的生产力。在他的文献列表中，1960年之前的条目有173条，20世纪60年代的有168条，70年代的有154条，80年代的有207条，90年代的有236条。此外，Simon博士还喜欢弹钢琴，尤其喜欢和演奏小提琴、中提琴及其他乐器的朋友合奏。他于2001年2月逝世，在此之前的几周，他还继续着自己的研究以及和学生们的交流。

语义网所表示的关系的类型决定了哪些问题是可以轻松解答的，哪些是更难解答的，哪些是不能解答的。例如，用图13-3所示的语义网，回答下列问题相当简单：

- Mary是学生吗？
- John的性别是什么？
- Mary住在宿舍还是公寓？
- Mary的学生ID是什么？

但是，回答下面的问题却很困难：

- 有多少女生，多少男生？
- 谁住在Dougherty堂？

注意，语义网中具有回答这些问题所必需的信息，只是不那么明显罢了。上面的两个问

题需要找到所有学生，但是不存在使这种信息一目了然的关系。这个网络是为表示学生个体与整个世界之间的关系而设计的。

这个网络不能回答下列问题，因为它没有表示必需的知识：

- John开的是什么牌子的车？
- Mary的眼睛是什么颜色？

我们知道Mary的眼睛具有一种颜色，因为她是学生，所有学生都是人，而所有人的眼睛都具有特定的颜色。只是根据网络中存储的信息，我们不知道她的眼睛究竟是什么颜色。

语义网是表示大量信息的强有力而通用的方式。难点在于建立正确的关系模型，用精确完整的数据填充整个网络。

13.2.2 检索树

第9章提到过一般的树结构，重点介绍了二叉树，即只有两个、一个或没有子女的树。一般的树结构的节点可能有多个子女，这种树在人工智能领域扮演着重要的角色。在对抗性情况（如博弈）中，树用于表示各种可能的选择。

检索树是表示游戏中所有可能的移动（包括你和你的对手的移动）的结构。你可以创建一个游戏程序，最大化它获胜的机会。在某些情况下，甚至可以保证它总是获胜。

> **检索树**（search tree）：表示对抗性情况（如博弈）中的所有选择的结构。

在人工智能领域使用的一般检索树中，一条路径表示玩家的一系列决定。每一层的决定说明了留给下一个玩家的选项。树中的每个节点表示一步移动，这个移动是以游戏中迄今为止已经发生的所有移动为基础的。

让我们定义一个简单的Nim游戏作为示例。在这个例子中，一行有一定数量的空格。第一位玩家可以在最左边的一组空格中放入一个、两个或三个X。然后第二个玩家可以紧接着X放入一个、两个或三个O。游戏就这样由两个玩家轮流继续下去。谁把自己的符号放入了最后一个（最右边的）空格，谁就获胜。

下面是Nim游戏的玩法示例，其中使用了9个空格。

```
初始状态：_ _ _ _ _ _ _ _ _
玩家1：   X X X _ _ _ _ _ _
玩家2：   X X X O _ _ _ _ _
玩家1：   X X X O X _ _ _ _
玩家2：   X X X O X O O _ _
玩家1：   X X X O X O O X X  玩家1获胜
```

图13-4所示的检索树展示了有5个空格（而不是上例中的9个空格）的Nim游戏的所有可能移动。在这个树的根节点中，所有格子初始时都是空的。接下来的一层展示了第一位玩家的三种选择（即放入一个、两个或三个X）。第三层根据第一位玩家所做的移动，展示了第二位玩家的所有可能选项。

注意，如果一次放入了大量的符号，那么下一个玩家的选项就会比较少，路径也会比较短。从根出发选择不同的路径移动，就可以知道每个玩家选择的选项。这个树表示了简化的游戏中的每个选项。

我们故意简化了Nim游戏，是为了可以展示一个简单的检索树。真正的Nim有许多重要的

不同之处，如其中有多个行，是从每行中删除项目，而不是添加项目，等等。不过，即使是简化的版本，也说明了一些有趣的数学思想。

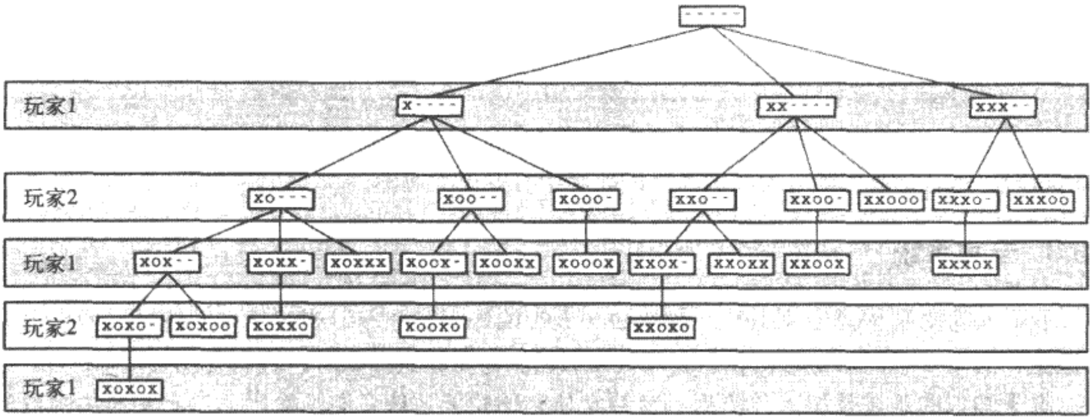


图13-4 Nim简化版的检索树

检索树分析的概念还适用于其他更复杂的游戏，如国际象棋。对于这种复杂的游戏，检索树要复杂得多，将具有许多节点和路径。考虑你在国际象棋游戏中第一步要做的所有可能的移动。然后考虑你的对手可能做出的所有反应。一个完整的国际象棋检索树包括每一层所有可能的移动。由于这样的树太大，所以即使具备现代的计算能力，在合理的时间限制内，也只能分析部分的树。

随着计算机变得越来越快，能够分析的部分检索树也越来越大，但仍然不能分析所有的分支。程序员在想方设法删减检索树，把那些人类玩家认为不合理的路径削减掉。不过，检索树仍然太大，不能进行完整的分析。

因此，问题变成了是采用**深度优先法**，总是先沿着可选的路径向下移动进行分析，还是选用**广度优先法**，先分析所有可能的路径而不向下移动。图13-5展示了这两种方法，它们可能都找不到关键可能性。这个问题在AI程序员之间争论了很多年，然而广度优先法趋向于生成最好的结果。一贯坚持无误的保守移动看来比偶尔采用惊人的移动效果好。大师级的下棋程序已经十分常见了。

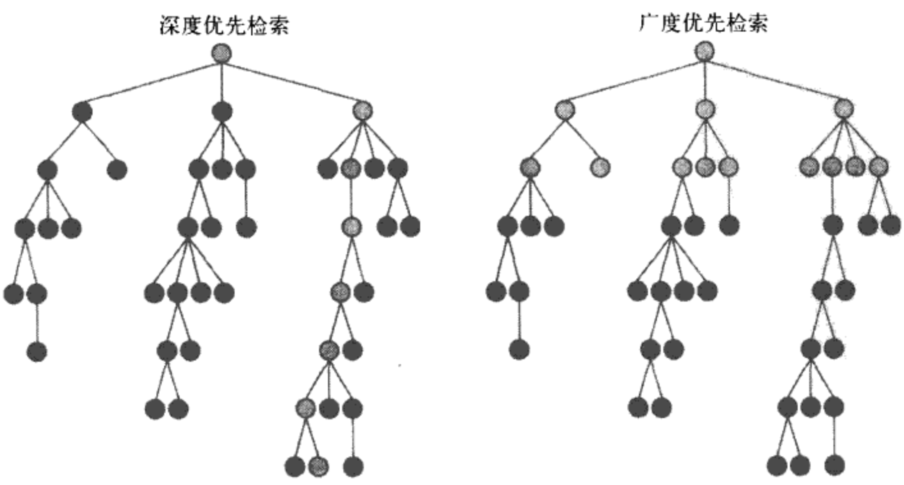


图13-5 深度优先检索和广度优先检索

> **深度优先检索**（depth-first approach）：优先沿着树的路径向下检索，而不是优先横向检索每层的检索法。
>
> **广度优先检索**（breadth-first approach）：优先横向检索树的每层，而不是优先向下检索特定路径的检索法。

1997年，IBM公司用专家系统开发的下棋程序深蓝在6局制的比赛中战胜了世界冠军Garry Kasparov。这是计算机第一次在大师级的比赛中打败人类冠军。

13.3 专家系统

我们常常需要依靠专家对特定领域独一无二的知识和理解。当健康有问题时，我们会去看医生；当车子不能启动时，我们会找机修工；当需要建造什么时，我们会找工程师。

基于知识的系统是嵌入并使用一套特定信息的软件系统，它从嵌入的信息集合中提取和处理特定的片段。术语**专家系统**和基于知识的系统一般是通用的，不过专家系统通常嵌入的是一个特定领域的知识，对这个领域的专业人员的专门技术进行了建模。当用户面临特定的问题时，会咨询专家系统，该系统将利用它的专门技术建议用户如何解决这个问题。

> **基于知识的系统**（knowledge-based system）：使用特定信息集合的软件。
>
> **专家系统**（expert system）：基于人类专家的知识构造的软件系统。

专家系统使用一套规则来指导处理，因此又叫做**基于规则的系统**。专家系统的规则集合又叫做它的*知识库*。**推理机**是专家系统的一部分，决定了如何执行规则，以及从而会得到什么结论。

> **基于规则的系统**（rule-based system）：基于一套if-then规则的软件系统。
>
> **推理机**（inference engine）：处理规则以得出结论的软件。

医生是一种活的专家系统。他们通过提问或化验收集信息。你的答案和化验结果可能会导致更多的问题和化验。医生知识库中的规则让他们知道接下来要问什么问题，然后他们用收集到的信息排除各种可能性，最终得出诊断结果。一旦识别出问题，他们就可以根据特定的知识提出适当的治疗方案。

让我们来演练一次专家系统的处理过程。假设你要问的是：我应该对草坪进行哪些修理？

嵌入了园丁知识的专家系统能够指导你如何做决定。我们来定义几个变量，以便在园丁系统中可以简化规则。

- NONE——这次不作任何修理。
- TURF——进行铺草皮修理。
- WEED——进行除草修理。
- BUG——进行除虫修理。
- FEED——进行施肥修理。
- WEEDFEED——进行除草和施肥修理。

这些值表示了专家系统在分析过当前情况之后可能得出的各种结论。接下来的布尔变量表示草坪当前的状态：

- BARE——草坪具有大块的空地。
- SPARSE——草坪普遍比较稀疏。
- WEEDS——草坪中有许多杂草。
- BUGS——草坪有虫子存在的迹象。

假设专家系统最初没有任何关于草坪状态的直接数据。例如，必须问用户草坪是否具有大块的空地。通过某些计算或其他数据库可以得到另一些专家系统能直接使用的数据，例如：

- LAST——最后一次修理草坪的日期。
- CURRENT——当前的日期。
- SEASON——当前的季节。

现在可以公式化一些系统用于推到结论的规则。这些规则采用if-then语句的形式。

```
if (CURRENT - LAST < 30) then NONE
if (SEASON = winter) then not BUGS
if (BARE) then TURF
if (SPARSE and not WEEDS) then FEED
if (BUGS and not SPARSE) then BUG
if (WEEDS and not SPARSE) then WEED
if (WEEDS and SPARSE) then WEEDFEED
```

注意，这只是这种系统中可能存在的规则的示例。真正的专家系统具有上千条规则以协助分析状况。即使对于我们使用的小例子，这套规则也没有覆盖所有的状况。

在执行过程中，推理机将选择一条规则，确定它是否可行。只需要向用户提问也许就能确定规则的可行性。如果这条规则是可行的，那么它可能会影响到其他规则的可行性。推理机将继续应用规则，直到没有可行的规则为止。不要把规则想成线性的（按序排列的）；推理机将应用所有能用的规则，反复循环，直到得出结论。推理机的运行可能会生成下列的交互过程：

系统：草坪有大块的空地吗？
用户：没有
系统：草坪有虫子的迹象吗？
用户：没有
系统：草坪普遍比较稀疏吗？
用户：是的
系统：草坪中有大量杂草吗？
用户：是的
系统：你应该进行除草和施肥修理。

注意，专家系统不会对任何它可以查到的信息提问，如最后一次修理的日期。显然我们的情况不是发生在冬天的，因为系统就潜在的虫害进行了提问。如果是冬天，虫害问题应该被省略。

比起其他的建议技术，专家系统具有许多优点。首先，它是面向目标的。它的重点不是抽象信息或理论信息，而是如何解决特定的问题。其次，它非常有效。它将记录之前的反应，

不会问无关的问题。最后，即使你不知道某些问题的答案，一个真正的专家系统也会通过精心构造的规则集合提供有用的指示。

> **LISP是AI使用的语言**
>
> LISP（LISt Processor）通常被看作AI的程序设计语言。John McCarthy于20世纪50年代末期为AI应用程序公式化了LISP。LISP的基本数据结构是一种有序的元素序列，叫做*列表*。其中的元素可以是单独的实体，也可以是其他列表。从专家规则到计算机程序，从思维处理到系统构件，列表几乎可以用于表示所有事物。LISP程序使用的是递归而不是循环。LISP和它的语系都属于函数范型。[1]

> **家里有PKC吗？**
>
> 医生为医疗诊断开发了一种专家系统，叫做问题诊断配合器（Problem Knowledge Coupler, PKC）。PKC并不像一些医疗网站那样，列出疾病的一些症状，然后让医生选择最合适的治疗方法。相反，PKC要求医生回答一系列有关病人症状的问题，然后列出最可能的诊断结果，并给出检查方法。PKC可以帮助医生用相关的医疗知识匹配病人的症状模式，然后识别出疾病并对症下药。

13.4　神经网络

我们曾经说过，一些人工智能研究员着重研究人脑究竟如何工作，从而构造出以相同方式工作的计算设备。计算机中的**人工神经网络**就是在尝试模拟人体神经网络的动作。让我们首先来看看生物神经网络是如何工作的。

> **人工神经网络**（artificial neural network）：尝试模拟人体神经网络的计算机知识表示法。

13.4.1　生物神经网络

神经元是传导基于化学的电信号的单个细胞。人脑包含数十亿个连接成网络的神经元。神经元在任何时刻都处于兴奋状态或抑制状态。处于兴奋状态的神经元将传导强信号，处于抑制状态的神经元则传导弱信号。一系列相连的神经元构成了一条路径。这条路径上的信号将根据它经过的神经元的状态被加强或减弱。一系列处于兴奋状态的神经元将创造出一条强信号路径。

生物神经元具有多个输入触角，叫做*树突*，一个主输出触角，叫做*轴突*。神经元的树突将接收来自其他神经元的轴突的信号，从而构成了神经网络。轴突和树突之间的空隙叫做*神经键*。如图13-6所示。神经键的化学结构调节了输入信号的强度。神经元的轴突上的输出是所有输入信号的函数。

神经元可以接受多个输入信号，然后根据相应的神经键给予每个信号的重要性控制它们的强度。如果有足够多的加权输入信号是强信号，那么神经元就进入兴奋状态，生成一个强输出信号。如果足够多的加权输入信号是弱信号，或者被该信号的神经键的加权因子削弱了，那么神经元将进入抑制状态，生成一个弱输出信号。

神经元每秒要跳动1000次，因此神经网络路径中的流量是稳定的。大脑的活动会使某些路径的信号加强了，某些路径的信号减弱了。在我们学习新事物时，大脑中将构成新的强神经路径。

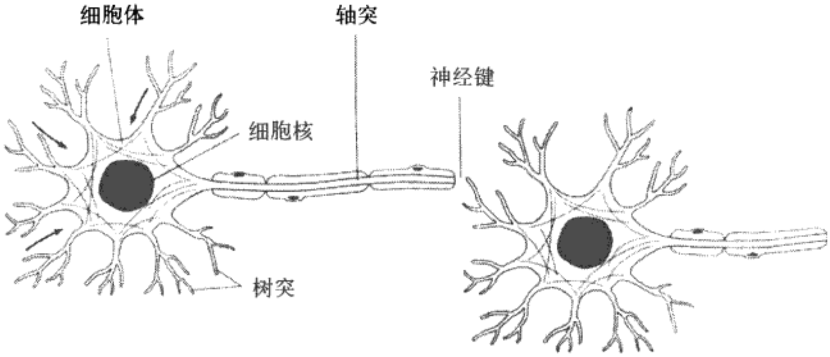


图13-6 生物神经元

13.4.2 人工神经网络

人工神经网络中的每个处理元素都类似于一个生物神经元。一个元素接收一定数量的输入值，生成一个输出值0或1。这些输入值来自于网络中的其他元素的输出，因此输入值也只会是0或1。每个输入值有一个数字权。元素的**有效权**是所有输入值与它的权的乘积之和。

> **有效权**（effective weight）：人工神经元中输入值和相应的权的乘积之和。

假设一个人工神经元接收的三个输入是v1、v2和v3。每个输入值的权分别是w1、w2和w3。那么它的有效权就是：

$$v1*w1+v2*w2+v3*w3$$

每个元素都有一个数字阈值，元素的有效权将与阈值进行比较。如果有效权大于阈值，这个元素将生成1。如果有效权小于等于阈值，该元素将生成0。

这种处理方法严密地反映出了生物神经元的活动。输入值对应于树突传入的信号。权对应于神经键对输入信号的控制效果。阈值的计算和用途对应于如果有足够的加权输入信号是强信号，则生成强信号的神经元。

让我们看一个真实的例子。在这个例子中，假设处理元素有四个输入值，因此有四个相应的加权因子。假设输入值是1、1、0和0，相应的加权因子是4、－2、－5和－2，该元素的阈值是4。有效权是：

$$1(4)+1(-2)+0(-5)+0(-2)=2$$

由于有效权小于阈值，所以该元素的输出是0。

虽然输入值只能是0或1，但是权却可以是任何值，甚至可以是负数。这里我们用整数作为权和阈值，不过它们也可以是实数。

每个元素的输出都是所有输入值的函数。如果输入是0，那么它的权就无关紧要。如果输入是1，那么权的数值无论是正数还是负数，都对有效权有很大影响。无论计算出的有效权是什么，它都要与元素的阈值进行比较。也就是说，有效权是15，对一个元素来说，足够生成输出为1，但对另一个元素来说，则会生成输出0。

人工神经网络中建立的一条路径是每个处理元素的函数。每个处理元素的输出将根据输

入信号、权和阈值而改变。不过输入信号只是其他元素的输出信号，因此改变单个处理元素的权和阈值就可以影响神经网络的处理。

调整神经网络中的权和阈值的过程叫做**训练**。训练神经网络，可以使它生成任何需要的结果。初始时，神经网络的权、阈值和初始输入通常都是随机产生的。这样生成的结果将与想要的结果相比，然后再调整其中的各个值。这一过程将持续到实现了想要的结果。

> **训练**（training）：调整神经网络中的权和阈值以实现想要的结果的过程。

考虑一下本章开头提出的问题：找出一幅相片中的猫。假设用神经网络解决这个问题，每个像素对应一个输出值。我们的目标是为属于猫的图像的每个像素生成输出1，为其他像素生成0。这个神经网络的输入值是像素的颜色表示。我们用多个包含猫的图像训练这个神经网络，使权和阈值逐渐接近想要的（正确的）输出。

想想看这个问题多么复杂！猫的形状、大小和颜色各不相同，而且它们出现在图像中的方向也千奇百怪，可能混在背景中，也可能没有混入背景。处理这个问题的神经网络难以置信的大，要将所有情况考虑在内。对神经网络的训练越多，它生成精确结果的机会就越大。

那么神经网络还适合做什么呢？它们已经被成功地应用于上千个应用领域，如商业和科学工作。它们可以用来决定是否可以贷款给某个客户，也可以用于光学字符识别，使计算机能够读入印刷文档，甚至还可以用在机场，探测行李箱中的塑胶炸弹。

神经网络之所以具有这样的通用性，是因为网络的权和阈值没有任何内在含义。它们的含义来自于我们施加给它们的解释。

13.5 自然语言处理

在科幻电影中，常常可以看到人们在和计算机交谈。宇宙飞船的船长可能会说："计算机，具有能治疗Laharman综合症的医疗设备的最近空间站是哪个。"计算机可能会回答道："42号空间站，距离14.7光年，具有必需的设备。"

科幻电影和现实的差距有多远呢？先不提星际旅行和高级药物，为什么我们还不能与计算机交谈呢？其实，我们可以在有限的程度上与计算机交谈。虽然我们还不能做到流畅的口头交流，但仍然取得了一定进展。有些计算机经过设置，可以对特定的口头命令做出响应。

要进一步探讨这个问题，首先必须认识人机语音交互过程中的三种基本处理类型。

- **语音识别**——识别人类所讲的话。
- **自然语言理解**——解释人类传达的信息。
- **语音合成**——再现人类的语音。

> **语音识别**（voice recognition）：用计算机来识别人类所讲的话。
> **自然语言理解**（natural language comprehension）：用计算机对人类传达的信息做出合理的解释。
> **语音合成**（voice synthesis）：用计算机制造出人类的语音。

计算机首先必须识别出独立的单词，然后理解这些单词的含义，最后（确定了答案后）生成组成响应的单词。

这些问题的共同点在于针对的都是**自然语言**，即人们用于交流的各种语言，如英语、波斯语或俄语等。自然语言固有的语法不规则性和二义性，使得处理它们具有很大的挑战性。

> **自然语言**（natural language）：人们用于交流的语言，如英语。

计算技术在这些领域已经取得了很大的进展。让我们来逐个地详细探讨这些问题。

13.5.1 语音合成

语音合成是个很好理解的问题，它有两种基本的解决方法——动态语音生成和录制语音。

采用动态语音生成法生成语音输出，计算机要分析构成单词的字母，生成这些字母对应的声音序列。人类的语音可以被划分成特定的声音单元——**音素**。图13-7展示的是美国英语的音素。

> **音素**（phonemes）：任何指定的语言中的基本声音单元的集合。

辅音			
符号	示例	符号	示例
p	pipe	k	kick, cat
b	babe	g	get
m	maim	ŋ	sing
f	fee, phone, rough	š	shoe, ash, sugar
v	vie, love	ž	measure
θ	thin, bath	č	chat, batch
ð	the, bathe	ǰ	jaw, judge, gin
t	tea, beat	d	day, bad
n	nine	ʔ	uh uh
l	law, ball	s	see, less, city
r	run, bar	z	zoo, booze

元音	
符号	示例
i	eel, sea, see
ɪ	ill, bill
e	ale, aim, day
ɛ	elk, bet, bear
æ	at, mat
u	due, new, zoo
ʊ	book, sugar
o	own, no, know
ɔ	aw, crawl, law, dog
a	hot, bar, dart
ə	sir, nerd, bird
ʌ	cut, bun

半元音	
w	we
h	he
j	you, beyond

双元音	
aj	bite, fight
aw	out, cow
ɔj	boy, boil

图13-7 美国英语的音素

选中合适的音素后，计算机将根据使用这个音素的上下文修改它的音调。此外还要确定每个音素的持续时间。最后，计算机要把所有音素组合在一起形成独立的单词。声音本身是通过电子方式生成的，模拟了人类声带的发声方式。

这种方法的难点在于不同人的发声方式不同，而且控制字符在每个单词的发音中所占的分量的规则也不一致。动态语音生成系统生成的语音，虽然每个单词都可以听懂，但是通常听起来都很机械、不自然。

另一种语音合成方法是对人声进行数字录音。语句是把单词按照适当的顺序排列得到的。有时，常用的短语或一组总是一起使用的单词会被录制为一个实体。电话语音邮件系统通常采用这种方法："要给Alex Wakefield留言，请按1。"

注意，每个单词或短语都要单独录制。此外，由于单词在不同的上下文中发音不同，所

以有些单词要录制多次。例如，问句结尾的单词比用在句中时音调高。对灵活性的要求越高，录制语音解决方案的难度就越大。

虽然动态语音生成技术一般不能生成真实的人声，但是它能发出每个单词的声音。录音回放功能提供的语音更真实；它使用的是真正的人声，不过它的词汇量仅限于预先录制好的单词。通常在使用的单词量比较小时才使用录音回放功能。

13.5.2 语音识别

在交谈的过程中，由于你可能不理解别人在讲什么，所以可能需要重复某些语句。并不是说你不理解别人言辞的含义，而只是说你不知道别人说了什么。发生这种情况有几种原因。

首先，每个人的发音不同。我们每个人的嘴、舌头、喉咙和鼻腔的形状都不同，它们影响了我们发音的语调和共振。因此，我们可以说“识别”出了某人的声音，就是从他的发音方式认出了他。不过，这还意味着每个人对指定单词的发音都不同，这就大大复杂化了识别单词的任务。口吃、喃喃自语、音量、方言和发声者的健康状况进一步复杂化了这个问题。

此外，人们是以连贯流畅的方式讲话的。单词排列起来构成了句子。有时，我们说得太快，以至于两个单词听起来像一个。人们具有把一系列发音分割成单词的能力，但是如果讲话的人说得过快，我们甚至都会听不明白。

与之相关的问题是单词自身的发音。有时，很难区分“ice cream”和“I scream”这两个短语。而同音异字词（如“I”和“eye”以及“see”和“sea”）的发音完全一样，但却是不同的单词。人们通常可以根据语句的上下文澄清这种情况，但这种处理需要更深的理解。

因此，如果连人偶尔都会遇到不能理解他人言语的问题，可想而知，这个问题对计算机来说有多难了。现代的语音识别系统仍然难以处理连续的交谈。最成功的系统采用的是不连贯的语音，其中每个单词都被明确地分割了出来。

当训练语音系统来识别特定的人声和单词集合后，语音识别取得了更大的进展。语音可以被录制为**声波纹**，绘制了讲特定单词时声音频率的变化。训练语音识别系统时，由一个人多次重复一个单词，使计算机记录下这个人对这个单词发音的平均声波纹。此后，将用所讲的单词与记录的声波纹进行比较，以确定这个单词是什么。

没有经过特定声音和单词训练的语音识别系统将与通用的声波纹比较以识别单词。虽然精确性差了一点，但使用通用声波纹可以避免耗时的训练过程，而且使任何人都可以使用语音识别系统。

> **声波纹**（voiceprint）：表示人声随着时间推移的频率变化的图。

13.5.3 自然语言理解

即使计算机能够识别人们所讲的单词，要理解这些单词的意思完全是另外一个任务。这也是自然语言处理最具挑战性的一个方面。自然语言固有二义性，也就是说，同样的语法结构，可能有多种有效的解释。产生这种二义性的原因有几种。

问题之一是一个单词可能有多种定义，甚至可以表示语言的多个部分。例如，单词“light”既可以是名词，也可以是动词。这种二义性叫做**词法二义性**。如果计算机想给语句附加含义，就要确定如何使用其中的单词。请考虑下面的句子：

Time flies like an arrow.（光阴似箭。）

> **词法二义性**（lexical ambiguity）：由于单词具有多种含义而造成的二义性。

这个句子的意思是时光流逝的速度就像射出去的箭一样快。这可能是你读到这个句子时的解释。但要注意，单词time还可以是动词，如给参加赛跑的运动员计时。而单词flies还可以是名词。因此这个句子还可以解释为一条指示，即“让时间像箭头记录飞行时间一样飞逝”。由于箭头不会对任何事物计时，所以你不会采用这种解释。但对别人来说，它同样有效。根据这些单词的定义，计算机不能判断出哪个解释是正确的。注意，这个句子甚至还有第三种解释法，即说明了一种特殊物种“time fly”的爱好是“arrow”，就像“fruit fly”（果蝇）喜欢香蕉一样。这种解释对你来说听起来很荒诞，但是对计算机来说，这些二义性给它理解自然语言带来了很大的麻烦。

自然语言的句子还会有**句法二义性**，因为短语的组合方式不止一种。例如：

I saw the Grand Canyon flying to New York.（在飞往纽约的途中我看到了大峡谷。）

由于峡谷不会飞，所以这个句子只有一种符合逻辑的解释。但是这个句子的结构却给了它两种有效的解释。要得到想要的结论，计算机必须知道峡谷是不会飞的，将这点考虑在内。

使用代词时，可能会发生**指代二义性**。考虑下面的句子：

The brick fell on the computer but it is not broken.（砖头落在了计算机上，但是它却没有被砸坏。）

什么没有被砸坏，砖头还是计算机？在这个例子中，我们可以假设代词“它”指代的是计算机，但这未必是正确的解释。事实上，如果是个花瓶落在了计算机上，那么在没有其他信息的情况下，即使是我们人类也未必判断得出“它”指代的是什么。

自然语言理解是一个很大的研究领域，远远超出了本书所能涵盖的范围。不过，理解为什么这一领域极具挑战性是相当重要的。

> **句法二义性**（syntactic ambiguity）：由于句子的构造方式有多种而造成的二义性。
> **指代二义性**（referential ambiguity）：由于代词可以指代多个对象而造成的二义性。

13.6 机器人学

机器人是我们所熟知的。从电视广告中的机器狗，到午夜新闻中的太空探索，到制造啤酒、汽车或装饰品的装配线，机器人已经成了现代社会的一部分。机器人学是研究机器人的科学，可以把机器人分为两大类——固定机器人和可移动机器人。你在装配线上看到的就是固定机器人，这些机器被固定在装配线上，产品从它们下面经过。由于固定机器人的世界非常有限，所以它的任务就内置在硬件上。因此，固定机器人几乎都应用于工业工程的领域。而可移动机器人就可以到处移动，必须与周围的环境进行交互。为可移动机器人的世界建模需要使用人工智能的技术。

13.6.1 感知–规划–执行范型

可移动机器人学研究的是能相对于环境移动并具有一定自治能力的机器人。为可移动机

器人的世界建模的原始方法利用了规划。规划系统是一种大型的软件系统，它能够根据给定的目标、起点和结局生成有限的动作集合（一套规划），如果（通常由人）执行这套动作，将实现预期的结果。这种规划系统综合了大量的领域知识，可以解决一般的问题。对于可移动机器人来说，领域知识是来自机器人的传感器的输入。采用这种方法，机器人的世界将被表示成复杂的语义网，机器人上的传感器将用于捕捉数据，构建网络。即使是简单的传感器，组装这种网络也很耗时；如果传感器是照相机，那么处理过程将极其耗时。这种方法叫做感知－规划－执行（sense-plan-act，SPA）范型，[2]如图13-8所示。



图13-8 感知－规划－执行（SPA）范型

传感器数据由世界模型解释，然后生成一个动作规划。机器人的控制系统（硬件）将执行规划中的步骤。一旦机器人移动，它的传感器就会得到新的信息，这个周期会反复执行，把新的数据引入语义网。如果处理新的传感器数据的速度不能满足它的使用速度，就会发生问题。（在世界模型识别出光照亮度的变化是由一个洞而不是阴影引起的之前，机器人可能已经掉进洞里了。）这种方法的缺陷在于机器人世界在一般系统中被表示成了领域知识，这种表示法太笼统，太宽泛，不适用于机器人的任务。

早期机器翻译的承诺并未实现

20世纪50年代末和60年代初，美国国家科学基金会（National Science Foundation）和美国军队都为大型机器翻译项目提供了基金。用计算机把一种自然语音翻译成另一种自然语音的幻想看来即刻就能变为现实。但是，使用大量的双语词典和逐字翻译的方法被证明是不足以实现这种翻译功能的。一个经典的故事就暴露了这种不足，即把英语"The spirit is willing, but the flesh is weak."（精神上乐意接受，但肉体上却很虚弱。）翻译成俄语，就成了"The vodka is acceptable, but the meat has spoiled."（伏特加酒是可以接受的，但肉已经坏了。）

研究员们意识到，在解决自动语言翻译的问题前，还需要对语言理解做更多的基础研究。

ELIZA

ELIZA是一个早期的自然语言程序。在下面的脚本中，该程序扮演一个精神治疗师。计算机的响应都用大写字母表示。

Men are all alike.（男人都一样。）

IN WHAT WAY?（哪些方面？）

They're always bugging us about something or another.（他们总是用这种事或那种事来烦我们。）

CAN YOU THINK OF A SPECIFIC EXAMPLE?（能给出一个具体的例子吗？）

Well, my boyfriend made me come here.（我的男朋友让我来这里。）

YOUR BOYFRIEND MADE YOU COME HERE?（你的男朋友让你来这里？）

He says I'm depressed much of the time.（他说我经常情绪低落。）

I AM SORRY TO HEAR YOU ARE DEPRESSED.（很遗憾听到你情绪不好。）

It's true. I am unhappy.（是真的，我觉得不开心。）

DO YOU THINK COMING HERE WILL HELP YOU NOT TO BE UNHAPPY?（你认为到这里来可以帮助你摆脱不开心的感觉吗？）

I need some help, that much seems certain.（显而易见，我需要帮助。）

...

WHAT ELSE COMES TO MIND WHEN YOU THINK OF YOUR FATHER?（当你想到自己父亲时还会想到什么？）

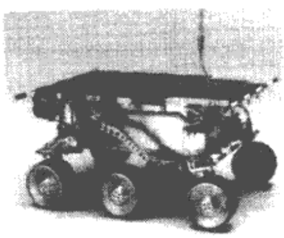
Bullies.（恶棍。）

DOES THAT HAVE ANYTHING TO DO WITH THE FACT THAT YOUR BOYFRIEND MADE YOU COME HERE?（这与你的男朋友要你来这里有什么关系吗？）[3]

在本书的网站上可以找到更多有关ELIZA程序的信息。

Sojourner探测器

Sojourner是人们第一次尝试远程控制位于另一个星球上的汽车。当飞船着陆之后，Sojourner沿着安装在登陆器踏板上的一个悬梯开了出来。地球上几千万个发烧友观看到了这个出场和此后的一系列探索工作。这个任务是要在一个火星日之内在登陆器和地球上的操作员之间传递信息。Sojourner能够以有指导的自控形式执行她的任务，这种形式将把目标位置（沿途停车点）或移动命令提前发给探测器，然后Sojourner将自行导航，安全移动到这些位置。[4]



由NASA/JPL/Caltech提供

Aibo迷悼念Aibo

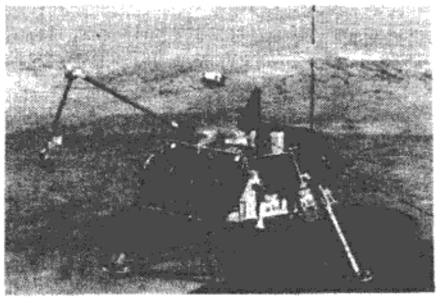
Sony公司沉痛地宣布了Aibo的逝世，Aibo这只机器狗能够学习自己主人的名字、发怒（眼睛变成红色的）以及表达开心之情（眼睛变成绿色的）。它的大小与玩具狗一样，销量达到了150 000多只。



由Sony Electronics公司提供

NASA发射了一对孪生机器人

2003年7月，NASA向火星发射了一对孪生机器人。自从到达火星，Spirit和Opportunity兄弟两个就开始不停地工作，以帮助科学家更好地了解这个红色行星。这对机器人的第三次延期返回日期是2006年9月。它们还在那里不停地发回着数据吗？



由NASA/JPL/Caltech提供

13.6.2 包孕体系结构

1986年，Brooks引入了*包孕体系结构*的概念，从而使机器人学中的范型发生了转变。[5]新的范型不再一次模拟整个机器人世界，而是赋予机器人一套简单的行为，每种行为与它所必需的一部分机器人世界关联在一起。除非这些行为有冲突，否则它们可以并行运行，在有冲突的情况下，每种行为要达到的目标的顺序决定了下一条要执行的是什么行为。这种体系结构的名字来源于行为的目标是可以排序的，或者说一种行为的目标包含在另一种行为的目标中。如图13-9所示。

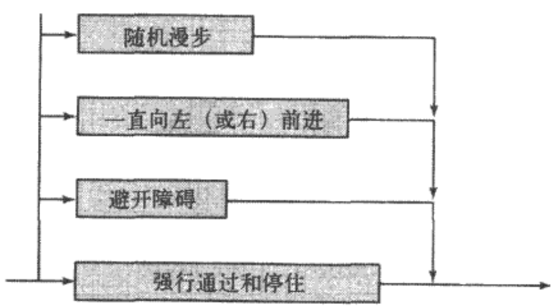


图13-9 新的控制范型

机器人胡须

研究人员开发出了一种有感觉的机器人，它采用模拟老鼠胡须的传感器。几种胡须都被尝试过，包括塑料线和人的头发。这种胡须贴在麦克风的振动膜上。在机器人遇到物体时，它会使麦克风的振动膜发生适当的变形。研究人员还在继续用各种胡须的组合和设计进行实验，以发现最好的配置。把触觉与其他感觉（如视觉）结合在一起的机器人会对手动任务自动化产生重大的影响。

当离一个对象太近时，“避开障碍”行为具有优先权，否则“一直向左前进”行为具有优先权。采用这种方法建模的机器人可以在房间中漫步几个小时而不会撞到任何对象或移动中的人。

Isaac Asimov定义的机器人学的三条定律完全适用于这种包孕体系结构。[6]如图13-10所示。

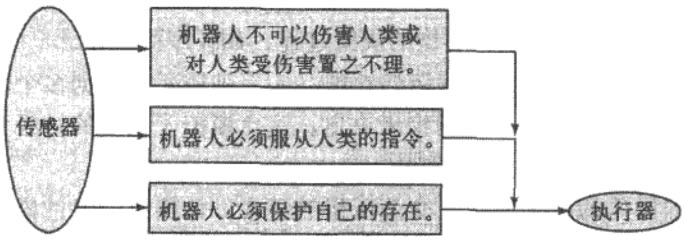


图13-10 Asimov的机器人学定律

另一种转变是把机器人的世界看作一个统一的坐标格，每个单元格表示等量的真实空间，整个世界是一个拓扑地图。拓扑地图把空间看作一幅由弧连接的地点图，具有相近和顺序的概念，但没有距离的概念。机器人可以局部地从一个地点移动到另一个地点，从而最小化了出错的机会。此外，在内存中表示拓扑地图比表示统一的坐标格更有效。

20世纪90年代，一种改进的方法流行了起来，它把规划与分布式世界中的一组行为结合了起来。

进入遥远的红色未知空间

认为用飞镖或者箭射中牛眼非常困难吗？试试看把两吨重的宇宙飞船发送到3.1亿英里之外的火星轨道上。2006年3月，NASA的火星探测轨道飞行器（Mars Reconnaissance Orbiter，MRO）开始围绕火星运行，它被誉为火星大气层之上成像最美的到达者。宇宙飞船自距离这个行星表面将近200英里的地方发送回NASA数据。它配备有望远镜式相机和帮助判断在行星表面之下是否有水或冰的证据的雷达系统。

13.6.3 物理部件

我们已经讨论过使机器人展示出类似于人类行为的各种方法，但是却忽略了机器人的物理部件。机器人是由传感器、执行器和计算部件（一个微处理器）构成的。传感器负责收集周围的数据，执行器负责移动机器人，计算部件负责给执行器发送指令。传感器是一种转换器，可以把物理现象转换成微处理器能够处理的电信号。有些传感器可以指示有光、无光或光的强度。近红外线接近探测器、运动探测器和爆炸探测器都可以用作传感器。此外，照相机和麦克风也可以用作传感器。机器人移动所需的三种最常用的系统是轮子、履带和机械腿。

小结

人工智能处理的是人类思想的建模和应用。图灵测试是确定一台机器是否能像人一样思考的衡量方法，采用的方式是模拟人类对话。

AI学科有很多需要研究的问题。最基本的问题是如何用计算机有效地处理形式表示知识。语义网是知识的图形化表示法，它捕捉了对象在真实世界中的关系。根据网络图的分析可以回答问题。检索树是表示对抗性移动（如比赛）的知识的重要方法。复杂的游戏（如国际象

棋）的检索树非常大，因此要有效地分析这种结构，还需要提出新的策略。

专家系统嵌入了人类专家的知识。它采用一套规则来定义条件，在这种条件下，可以得出某些结论。专家系统适用于多种类型的决策过程，如医疗诊断等。

人工神经网络模拟了人脑神经网络的处理。人工神经元将根据多个输入信号生成一个输出信号，输入信号的重要性由它们的权来决定。这点模拟了人类的神经元，即由神经键调节从一个神经元到下一个神经元的信号强度。

自然语言处理操作的是人们用来交流所用的语言，如英语。通过模拟人声的音素或重放预先录制的单词可以合成语音。在单词分离的情况下，可以最好地实现语音识别，训练系统识别特定人的音波纹也可以实现比较好的语音识别效果。所谓自然语言理解，就是给予谈话的内容一个解释，这是自然语言处理的核心。自然语言中存在的各种二义性（即一个句子有多种解释）大大复杂化了自然语言的理解。

机器人学是研究机器人的科学，它可以分为两大类——固定机器人学和移动机器人学。固定机器人是被固定起来，等待要处理的对象经过它们的机器人。移动机器人则能够移动，需要利用人工智能的技术对它们所处的环境进行建模。

道德问题：HIPAA（健康保险携带和责任法案）

无论是否与我的职业习惯相关，只要是我听到或者看到的，不应该公开谈论的事情，我决不会泄露，这些都是应该保密的。

——Hippocratic Oath

1996年通过了一个新法案——健康保险携带和责任法案（HIPAA），以鼓励用电子手段使用及发布病人的保密信息，并使之合法化。该法案允许更多人员在不经病人同意的情况下查阅病历。提出该法案的人声称，当前的技术能够收集和共享以前的保密信息，这样做可以帮助医生诊断疾病，帮助研究人员开发新的药物，帮助政府跟踪及对抗对群众的健康威胁（如2003年爆发的SARS）。虽然有这些好处，但是反对者们声称，无限制地访问病历中的所有信息，既不必要，也没有任何好处。他们引用了精神病学家的例子，保险公司威胁精神病学家，如果不向保险公司公开病人的病历，就将他们从保险业的被保险者中除名，精神病学家被迫允许保险公司的人员查阅所有病人的病历；医院把所有病历发送给支付保险费的第三方公司；HMO把这些个人信息提供给雇员。上述的每种行为都侵犯了病历隐私权。只需要敲几下键盘，商人就能获得你的处方清单，给你发送促销资料，或者把这些信息卖给另一方。1993年的法案允许构建的数据库包括服用Prozac的人的清单、堕胎的人的清单以及个人所患疾病的清单。政府机构和执法官员都能访问这些数据。

另一个与健康护理相关的技术争执是医疗身份证的使用。虽然这个计划是美国提出的，但近来却在英国实施了。公民们将得到一张磁卡。每次开处方或者看医生，都要刷卡，数据从而被收集起来。这样，每个人从出生开始的病历都可以用电子方式访问。此外，还有计划在这张卡上加入遗传信息，例如乳腺癌和心脏病的遗传标记。

疾病跟踪能够更有效地销售药物等，但是这些好处能抵消保密病历的好处吗？隐私权拥护者声称，上述这些专制的技术应用将导致激冷效应。如果知道自己与医生的关系不再是保密的，病人可能会拒绝就医，或者隐瞒敏感信息，从而降低了他们接受的治疗的效果。与HIPAA的斗争还在继续，结果明朗之前，可能还需要一段时间。同时，医疗隐私权成为了历史，这已经是事实了。

练习

为练习1～5中的例子找出匹配的二义性类型。

A. 词法二义性　　B. 指代二义性

C. 句法二义性

1. “Stand up for your flag.”
2. “Go down the street on the left.”
3. “He drove the car over the lawn mower, but it wasn't hurt.”
4. “I saw the movie flying to Houston.”
5. “Mary and Kay were playing until she came inside.”

判断练习6～21中的陈述的对错：

A. 对　　B. 错

6. 计算机执行某些任务比人类执行得好。
7. 人类执行某些任务比计算机执行得好。
8. 能够通过图灵测试的计算机系统可以看作是智能的。
9. 有些AI研究员认为，在计算机能够像人脑一样处理信息之前，我们不能实现真正的人工智能。
10. 语义网是用来对关系建模的。
11. 如果信息存储在语义网中，那么很容易回答与之有关的问题。
12. 在大师级的国际象棋比赛中，计算机从来没有赢过人类。
13. 推理机是基于规则的专家系统的一部分。
14. 生物神经元接受一个输入信号，生成多个输出信号。
15. 人工神经网络中的每个元素都有数字加权。
16. 语音合成是自然语言处理中最难的部分。
17. 每个人的声波纹都是唯一的，可以用于训练语音识别系统。
18. 计算机对单词“light”有多种解释。
19. 句法二义性对自然语音理解来说不成问题。
20. 机器人是使用感知-规划-执行法来控制它的移动。
21. Isaac Asimov定义了机器人学的三条基本定律。

为练习22～30中的任务找出能最轻松解决它的对象。

A. 计算机　　B. 人

22. 识别图片中的一只狗。
23. 求100个四位数的和。
24. 解释一首诗。
25. 指纹匹配。
26. 绘制一幅风景画。
27. 进行谈话。
28. 学习发音。
29. 判断有罪或无罪。
30. 给予关爱。

练习31～76是问答题或简答题。

31. 什么是图灵测试？
32. 如何组织和管理图灵测试？
33. 什么是弱等价性，如何把它应用于图灵测试？
34. 什么是强等价性？
35. 什么是Loebner奖？
36. 列举并简单说明本章介绍的5个AI问题。
37. 列举并定义两种知识表示法。
38. 第9章定义的数据结构是如何用于表示语义网的？
39. 为你的家族成员之间的关系建立一个语义网。列出5个能用你的语义网轻松回答的问题，再列出5个稍有难度的能回答的问题。
40. 创建一个语义网，捕捉一篇报纸文章的一节中的信息。
41. 语义网借用了哪些面向对象的属性？
42. 什么是检索树？
43. 为什么复杂游戏（如下棋）的检索树都很大？
44. 请区别深度优先检索和广度优先检索。
45. 删减检索树是什么意思？
46. 请区别基于知识的系统和专家系统。
47. 请区别基于规则的系统和推理机。
48. 请举出一个人类专家的例子。
49. 模拟具有某个领域的专业知识的专家且基于知识的系统叫做什么？
50. 为什么专家系统又叫做基于规则的系统？
51. 专家系统中决定如何执行规则以及可以得出什么结论的软件部分叫做什么？
52. 在专家系统中如何表示规则？
53. 专家系统有哪些优点？
54. 传导基于化学的电信号的单个细胞叫什么？

55. 一系列相连的神经元可以构成什么？
56. 信号依靠什么在特定的路径中传输？
57. 生物神经元中的多个输入触角是什么？
58. 生物神经元的主输出触角是什么？
59. 一个神经元的树突从哪里接收来自另一个神经元的信号以构成网络？
60. 树突和轴突之间的空隙叫什么？
61. 调节神经键强度的是什么？
62. 神经键的作用是什么？
63. 在人工神经网络中如何模拟神经键？
64. 人工神经元的有效权是什么？
65. 如何计算一个人工神经元的输出值？
66. 如果人工神经网络中的一个处理元素接收了5个输入信号，分别为0、0、1、1和0，它们相应的权是5、－2、3、3和6，阈值是5，那么这个元素的输出是什么？
67. 如果人工神经网络中的一个处理元素接收了5个输入信号，分别为0、0、1、1和0，它们相应的权是5、－2、3、3和6，阈值是7，那么这个元素的输出是什么？
68. 什么是音素？
69. 描述两种实现语音合成的方式。
70. 哪些问题会影响识别人类语音中的单词的效果？
71. 如何训练语音识别系统？
72. 为什么个人化的语音识别系统比通用的系统好得多？
73. 列举并描述两种机器人。
74. 什么是规划系统？
75. 如何定义包孕体系结构？
76. 机器人是由什么组成的？

思考题

1. 如果你在图灵测试中担任质问者，请考虑5个你可能提出的问题。为什么计算机难于很好地解答这些问题？
2. 你认为强等价性是可能的吗？如何证明这一点？
3. 当你想到机器人，会想到些什么？是想到机器人在地板上疾走吗？还是生产软饮料或啤酒的装配线？
4. 如果你知道自己敏感的个人信息和医疗信息有可能泄露出去，会不会拒绝把它们留给你的医生呢？
5. 如果在美国广泛使用医疗身份证，那么应该在其中加入遗传信息吗？

第14章 模拟、图形学和其他应用程序

使用模型表示现象或情况的技术叫做模拟。飞机制造商通过建立风洞来研究新的飞行器设计中的机翼周围的气流。飞行模拟器是一种模型，可以再现飞行器对驾驶员所做的动作的反应，因此使驾驶员能够在进入驾驶舱之前学习控制飞行器，驾驶员要在飞行模拟器中花费大量的时间。在新的超级市场的规划方案定稿之前，可以运行一个计算机程序，根据预计的顾客流量，以协助确定需要多少个收银台。

这一章将介绍模拟背后的理论知识，并且研究几个具体的例子，包括预报天气的模型。然后介绍另外四种应用，即计算机图形学、嵌入式系统、电子商务和计算机安全，以此结束关于应用层的讨论。

目标

学完本章之后，你应该能够：

- 定义模拟。
- 举出复杂系统的例子。
- 区分连续事件模拟和离散事件模拟。
- 解释如何应用面向对象的设计原理构造模型。
- 列举并讨论排队系统的四个部分。
- 解释天气和地震模型的复杂性。
- 解释嵌入式系统的概念并举出家居生活中的嵌入式系统的例子。
- 描述图形图像生成中的重要主题。
- 解释与单一图像相比动画需要关注的更多问题。
- 定义并解释电子商务在当今社会中的角色。
- 列举三种鉴别凭证。
- 定义下列与计算机安全相关的术语：恶意代码、病毒、蠕虫、特洛伊木马、逻辑炸弹、哄骗、后门、缓存溢出、拒绝服务以及中间人。

14.1 什么是模拟

模拟是研究复杂系统的有力工具。所谓**模拟**，就是设计复杂系统的模型以及为观察结果而对其进行实验性操作。模型既可以是纯物理性的（如风洞），也可以是软件控制的物理对象（如太空船或飞行模拟器），还可以是纯逻辑性的（如用计算机程序表示的模型）。

> **模拟**（simulation）：设计复杂系统的模型并为观察结果而对该模型进行实验。

从20世纪50年代中期开始，计算机模拟就被用于协助决策。复杂系统的计算机模型使决策者们能够逐渐了解系统的性能。一间银行需要多少出纳员？如果两个操作台之间的距离加大，那么生产线上的原料流会不会加快？明天的天气怎么样？哪里是设置新消防队的最佳地

点？通过模拟，我们可以对所有这些问题有相当多的了解。

14.1.1 复杂系统

*系统*是那种只能凭直觉理解，而很难定义的术语。字典给出了几种系统的定义，它们的共同之处即把系统定义为一组相互作用的对象，这些对象可以是有生命的，也可以是无生命的。一组软件和硬件就构成了计算机系统。一组铁轨和火车就构成了铁路运输系统。一组老师和学生就构成了学校系统。

最适合模拟的系统是*动态的*、*交互式的*和*结构复杂的*系统，[1]也就是说，这些系统应该是难于理解和分析的。动态系统的行为将随着时间而改变。这种行为变化的方式可以通过数学公式理解和捕捉，例如导弹通过非扰动的大气层的飞行距离。有些行为只能被部分理解，但却服从逻辑表示，如一个交通灯处到达的人流量。系统的定义暗示了其中的对象具有交互性，系统中的交互性越多，这个系统就越适合模拟。例如，处于空中交通控制下的飞机的行为。飞机自身的性能特征、与空中交通控制员的交流、天气状况以及由于地面问题引起的航线变化都对飞机的行为有影响。最后，系统应该由许多对象构成，否则模拟它就是浪费时间。

14.1.2 模型

*模型*是另一个能够理解但难于定义的术语。字典中有两种定义与模拟有关，其一即一种类比，用于帮助可视化某些不能直接观察到的东西，其二是一组由实体或事件状态的数学表达法表示的假设、数据和推理。虽然这两个定义看来有很大不同，但它们有一线共同之处，即模型都是某种事物的抽象。在第一种定义中，模型表示的是不能完全理解的事物，我们被迫说它像其他某种事物。在第二种定义中，对系统的理解足以用一组数学规则来描述它。

在模拟中，**模型**即真实系统的抽象。它是系统中的对象和管理对象相互作用的规则的表示法。这种表示法可以是具体的，如太空船和飞行模拟器，也可以是抽象的，如分析所需的收银台数量的计算机程序。在余下的有关模拟的讨论中，模型都是抽象的，实现都在计算机程序中。

模型（model）：真实系统的抽象，系统中的对象和管理对象相互作用的规则的表示法。

14.1.3 构造模型

构造模型的关键是确定一个足以描述被调查的行为的特征或特性的小集合。记住，模型是真实系统的抽象，而不是系统本身。因此，在能够精确描述系统行为的特征太多和太少之间有一条精密的分界线。我们的目标是构造一个能够描述相关行为的最简单模型。

有两种不同的模拟类型，为每种类型选择特征或特性集合的过程不同。这两种类型的区别在于表示时间的方式，一个采用的是连续变量，另一个采用的是离散事件。

连续模拟

连续模拟把时间看作连续的，用一组反映特征集合中的关系的微分方程表示时间的变化。因此，为系统建模而选择的特征或特性的行为必须是能够用数学表达的。例如，气象模型就属于这种类型。天气模型的特征包括风力、气温、湿度、云层、降水量，等等。可以用一组

偏微分方程对这些成分随着时间而产生的相互作用建模，这组方程能估量出这些成分在三维空间中的变化率。

由于连续模拟中的特征具有理论本质，所以工程师和经济学家们经常使用这种技术。在这些领域中，可用的特征和它们之间的相互作用已经为人熟知了。在后面的小节中，我们还会详细地介绍气象模拟。

离散事件模拟

离散事件模拟由实体、*属性*和*事件*构成。实体表示真实系统中必须明确定义的对象。也就是说，系统的特征或特性是对象。例如，如果要对一个制造厂建模，那么各种机器和要生产的产品就是实体。属性是一个特定实体的特征。标号、购买日期和维修历史是某部机器的属性。事件是实体之间的相互作用。例如，把一台机器的输出发送给下一台机器作为输入就是一个事件。

流经系统的对象通常被表示为实体。例如，一台机器的输出可以是传递给下一台机器的对象。未加工的制品流就这样从一台机器传送到另一台机器（一系列事件），最终将生成可爱的新产品。实体也可以表示其他实体所需的资源。例如，在银行模型中，出纳员就是一种资源。如果没有空闲的出纳员，客户就要排队等候，直到有出纳员为之服务为止。

构造一个好模型的关键是选择实体以表示系统，并正确地决定定义事件结果的规则。Parteto定律认为，在每个实体集合中，都有一些必需的实体和许多微不足道的实体。平均说来，一个系统约80%的行为都可以用20%的成分的动作解释。[2] 模拟定义的第二部分“为观察结果而对模型进行实验”给了我们从何着手的线索。要观察什么结果呢？这个问题的答案是确定模型中必须表示的系统实体的着手点。实体和定义实体相互作用的规则必须足以生成要观察的结果。

由于抽象模型是用计算机程序实现的，所以可以应用面向对象的设计来解决建模问题。模型中的实体即对象类。实体的属性即类的属性。那么事件相当于什么呢？事件就是实体的责任。定义实体相互作用的规则由类的协作表示。

下一节将把这些技术应用于具体的例子。

14.1.4　排队系统

让我们来看一种非常有用的模拟——*排队系统*。排队系统是一种离散事件模型，它使用随机数表示事件的到达和持续。排队系统由服务器和等待服务的对象队列构成。第9章介绍过，队列是先进先出的结构。我们的日常生活中时常会用到排队系统。当你在杂货店排队等候结账或者在银行提款时，使用的就是排队系统。当你向大型机提交了一个“批作业”（如编译）后，你的作业必须排队等候CPU完成它之前的各项作业。当你致电航空公司预订机票时，会听到这样的录音“谢谢您致电Air Busters。接线员忙，您的电话很快会被接听，请稍后。”此时你使用的还是排队系统。

请稍后

等待是个严重的问题。排队系统的目的是尽可能地完全利用服务器（出纳员、收款员、CPU、接线员，等等），使等待时间处于合理的限度。要实现这一目标，通常需要在花费和客户满意度之间进行折中。

从个人角度来说，没有人愿意排队。如果给超级市场中的每个顾客配备一个收银台，顾

客们一定非常乐意。但是这样超级市场的买卖就不可能做得太久。因此，折中的办法是根据超级市场的预算限制收银台的数量，同时一般不让顾客等候太久。

那么一个公司如何决定服务器数量和等待时间之间的最佳折中呢？一种方法是靠经验，即尝试使用不同数量的服务器，看多少服务器可以解决问题。这种方法有两个问题，即昂贵且耗时。另一种方法是使用计算机模拟。

要构造一个排队模型，必须知道四点：

1. 事件的数量以及它们如何影响系统，以确定实体相互作用的规则。
2. 服务器的数量。
3. 到达时间的分布情况，以确定是否把一个实体加入系统。
4. 预计的服务时间，以确定事件的持续时间。

模拟将使用这些特征来预测平均等待时间。可以改变服务器的数量、到达时间的分布情况和服务时间，从而分析等待时间以确定什么是合理的折中。

一个实例

考虑只有一个出纳员的免下车银行的例子。每辆车平均要等候多久？如果生意扩大，车辆到达得更加频繁了，对平均等待时间有什么影响？银行何时需要增设第二个免下车窗口？

这个问题具有排队模型的特征。实体是服务器（出纳员）、受服务的对象（车内的顾客）和等待服务的对象（车内的顾客）的队列。平均等待时间是观察的目标。这个系统中的事件是顾客的到达和离开。

SIMULA是为模拟而设计的

位于奥斯陆的挪威计算中心（Norwegian Computing Centre，NCC）的Ole-Johan Dahl和Kristen Nygaard在1962年到1967年间设计并构建了程序设计语言SIMULA，这是一种为离散事件模拟而设计并实现的语言。此后，SIMULA被扩展并重新实现成了一种完整的通用程序设计语言。虽然SIMULA从未得到过广泛的应用，但它对现代程序设计方法有着深远的影响。就是SIMULA引入了面向对象这种重要的语言构造，如类和对象、继承以及多态性。[3]

让我们看看用时间驱动的模拟如何解决这个问题。在时间驱动的模拟中，每隔固定的时间间隔（如1分钟）就观察一次模型。为了模拟时间单元（如1分钟）的流逝，我们累计时间，预定运行模拟的时间，如100分钟。（当然，模拟的时间通常比真正的时间流逝得快，100个模拟分钟在计算机中只是一闪而过。）

把模拟想象成一个大循环，每次循环为时钟的一个值（在我们的例子中是从1到100）执行一套规则。下面是循环主体处理的规则：

- 规则1：如果一个顾客到达了，他或她将进入队列。
- 规则2：如果出纳员空闲，而且有顾客在等待，那么队列中的第一个顾客将离开队列，前进到出纳员的窗口。开始为该顾客的服务计时。
- 规则3：如果顾客位于出纳员的窗口，那么这位顾客剩余的服务时间将减少。
- 规则4：如果有顾客在排队，那么要增加他们在队列中等待的时间记录。

这个模拟的输出是平均等待时间。我们用下面的公式计算这个值：

平均等待时间=所有顾客的总等待时间/顾客总数

银行可以根据这个输出判断他们的客户是否在某个出纳员系统之前等候太久了。如果判断结果是顾客等候过久，那么银行将增设出纳员。

还没有结束！还有两个问题没有解答。如何知道一个客户到达了？如何知道一个客户的服务结束了？为此，必须提供到达时间和服务时间的信息模拟。在模拟中用变量（参数）表示它们。我们不可能精确地预测顾客何时到达以及每个客户将占用多少服务时间，但是可以根据经验猜测例如每隔5分钟会有一个顾客到达，大多数顾客需要3分钟服务时间。

那么在这个计时单元系统中，如何知道一个作业是否到达了呢？答案是两个因子的函数，即顾客到达的时间间隔（这里是5分钟）和可能性之间的分钟数。*可能性*？排队模型是基于可能性的？不完全如此。让我们用另一种方式表示顾客到达的时间间隔，即每个指定的计时单元中作业到达的*概率*。概率的范围是0.0（没可能）到1.0（绝对的事情）。如果平均每隔5分钟到达一个新作业，那么任何分钟内有顾客到达的可能性是0.2（5个机会之一）。因此，在特定分钟内有顾客到达的概率是1除以到达间隔的分钟数。

那么如何表示这种可能性呢？在计算机术语中，可以用*随机数发生器*表示可能性。我们编写了一个生成0.0到1.0之间的随机数的函数来模拟顾客的到达，采用的规则如下：

1. 如果随机数在0.0和到达概率之间，说明作业已经到达了。
2. 如果随机数大于到达概率，那么在这个计时单元中，没有作业到达。

通过改变到达速率，可以模拟交易需要3分钟的单出纳员系统随着到达车辆的增多会出现哪些情况。我们也可以基于概率模拟服务时间的长度。例如，可以这样模拟，60%的顾客需要3分钟的服务时间，30%的人需要5分钟，另外10%需要10分钟。

模拟并没有给我们特定的答案，甚至不是一个答案。模拟只是尝试回答“假设”问题的一种方法。我们需要构造模型，然后运行模拟多次，尝试各种可能的参数组合，观察平均等待时间。如果一辆车到达得太快会出现什么情况？如果服务时间缩短10%会出现什么情况？如果增加了一个出纳员会出现什么情况？

其他类型的队列

前面的例子使用的队列是FIFO队列，即受到服务的实体是在队列中停留时间最久的实体。另一种队列是*优先队列*。在优先队列中，每个项目都有一个优先级。每次出列的项目都是优先级最高的项目。优先队列的操作就像电视秀M*A*S*H中的治疗类选法，当伤员到达后，医生会给每个病号一个标签，标明他的受伤严重程度。伤势最严重的伤员将优先进手术室。

还有一种排列事件的模式，采用了两个FIFO对象，一个用于较短的服务时间，一个用于较长的服务时间。这种模式有点像超级市场的快速通道，如果你要买的物品少于10件，就可以进入快速通道的队列，否则必须进入正常通道的队列。

Ivan Sutherland



Ivan Sutherland在学术、工业研究以及商业领域都受过专业训练。Sutherland在他的网页上列出的头衔包括工程师、企业家、资本家和教授。他曾获得过ACM颁发的图灵奖、Smithsonian计算机世界奖、美国国家工程院颁发的First Zworykin奖和Price Waterhouse Information Technology Leadership奖的终身成就奖。

Sutherland拥有Carnegie理工学院的学士学位、加州理工学院的

硕士学位和麻省理工学院的博士学位。他的博士论文《Sketchpad: A Man-machine Graphical Communications System》开创了用光笔在屏幕上直接绘图的先河。图形的模式可以存储在内存中，此后再像其他数据一样被读取和操作。画板是第一种GUI (Graphical User Interface，图形用户界面）设备，此时GUI这个术语还没有出现，它开创了计算机辅助设计（CAD）这个领域。

20世纪60年代初，美国国防部和国家安全局（National Security Agency，NSA）都是计算研究的先锋部队。Sutherland毕业后被劝导加入了陆军，分配到了NSA。1964年，他被调到了国防部的高级研究规划署（ARPA，以后的DARPA)，担任ARPA的信息处理技术办公室的主管，负责管理计算机科学的研究项目。

退出军队后，他在哈佛担任副教授。开发使人们能够用图像与计算机进行交互的设备——画板成了他的工作。他的目标是“最大显示”，即要包括全彩色和满足用户各种视角的立体显示。由于头盔式显示器（HMD）非常重，所以把理论变为现实比假想难得多。因此，最初的实现显示在墙上或天花板上，而不是头盔中，这使它有了个外号“达摩克利斯之剑”。

1968年，Sutherland迁到了犹他大学，继续从事HMD系统的研究。他和犹他大学的另一位教员David Evans共同创建了Evans & Sutherland公司，专卖模拟、训练和虚拟现实应用所需的可视系统的硬件和软件。1975年，Sutherland返回加州理工学院，担任计算机科学系的主任，在此，他协助在课程中引入了电路设计。

1980年，Sutherland离开了Caltech，建立了Sutherland, Sproull, and Associates，一家咨询和风险投资公司。他具有8项计算机制图和硬件方面的专利，并继续从事他在硬件技术方面的研究。现在，他是Sun Microsystems公司的副总裁和合伙人。

1988年，Sutherland的图灵奖获奖词是：

> 他以画板开创了计算机制图领域，并不断地提出具有远见的想法。虽然画板是25年前编写的，但他引入的许多技术即使在今天仍然非常重要，包括刷新屏幕用的显示文件、图形对象建模用的递归遍历的层级结构、几何变换用的递归方法和面向对象的程序设计风格。之后的创新还包括观察立体和彩色图像用的长柄望远镜、显示数字化图像的算法、剪裁多边形的算法和用隐线表示曲面的算法。

暂且不提Sutherland收到的各种荣誉，最近他宣布了最令自己骄傲的杰作是他的四个孙子。

14.1.5 气象模型

上一节用离散输入和输出进行了相当简单的模拟。下面将讨论的是一种连续模拟——预测天气。天气预测的细节只有专业气象学家才知道。一般说来，气象模型是以由时间决定的流体力学和热力学的偏微分方程为基础的，这些方程的变量包括两个水平风速、垂直风速、气温、气压和水汽浓度。图14-1展示了一些方程。不要担心，这些方程的使用超出了本书的范围，我们只是想说明这些类型的模型中有些复杂的处理。

为了预测天气，首先输入观察得来的这些变量的初始值，然后求积分得到这些变量的值。[4]用预计的值作为初始条件，再次求这些方程的积分。用上一次求积分的预计值作为当前积分的观察值，可以不时给出天气预测。这些方程描述的都是实体在模型中的变化率，因此每个

解决方案的答案给出的值可以用于预测下一套值。

水平分动量：

$$\frac{\partial p^*u}{\partial t}=-m^2\left[\frac{\partial p^*uu/m}{\partial x}+\frac{\partial p^*vu/m}{\partial y}\right]-\frac{\partial p^*u\sigma}{\partial\sigma}+uDIV$$

$$-\frac{mp^*}{\rho}\left[\frac{\partial p'}{\partial x}-\frac{\sigma}{p^*}\frac{\partial p^*}{\partial x}\frac{\partial p'}{\partial\sigma}\right]-p^*fv+D_u$$

$$\frac{\partial p^*v}{\partial t}=-m^2\left[\frac{\partial p^*uv/m}{\partial x}+\frac{\partial p^*vv/m}{\partial y}\right]-\frac{\partial p^*v\sigma}{\partial\sigma}+vDIV$$

$$-\frac{mp^*}{\rho}\left[\frac{\partial p'}{\partial y}-\frac{\sigma}{p^*}\frac{\partial p^*}{\partial y}\frac{\partial p'}{\partial\sigma}\right]-p^*fu+D_v$$

垂直分动量：

$$\frac{\partial p^*w}{\partial t}=-m^2\left[\frac{\partial p^*uv/m}{\partial x}+\frac{\partial p^*vw/m}{\partial y}\right]-\frac{\partial p^*w\sigma}{\partial\sigma}+wDIV$$

$$+p^*g\frac{p_0}{\rho}\left[\frac{1}{p^*}\frac{\partial p'}{\partial\sigma}+\frac{T_v'}{T}-\frac{T_0p'}{Tp_0}\right]-p^*g\left[(q_c+q_r)\right]+D_w$$

气压：

$$\frac{\partial p^*p'}{\partial t}=-m^2\left[\frac{\partial p^*up'/m}{\partial x}+\frac{\partial p^*vp'/m}{\partial y}\right]-\frac{\partial p^*p'\sigma}{\partial\sigma}+p'DIV$$

$$-m^2p^*\gamma p\left[\frac{\partial u/m}{\partial x}-\frac{\sigma}{mp^*}\frac{\partial p^*}{\partial x}\frac{\partial u}{\partial\sigma}+\frac{\partial v/m}{\partial y}-\frac{\sigma}{mp^*}\frac{\partial p^*}{\partial y}\frac{\partial v}{\partial\sigma}\right]$$

$$+p_0g\gamma p\frac{\partial w}{\partial\sigma}+p^*p_{0gw}$$

气温：

$$\frac{\partial p^*T}{\partial t}=-m^2\left[\frac{\partial p^*uT/m}{\partial x}+\frac{\partial p^*vT/m}{\partial y}\right]-\frac{\partial p^*T\sigma}{\partial\sigma}+T\,DIV$$

$$+\frac{1}{\rho c_p}\left[p^*\frac{Dp'}{Dt}-p_0gp^*w-D_{p'}\right]+p^*\frac{Q}{c_p}+D_T,$$

其中

$$DIV=m^2\left[\frac{\partial p^*u/m}{\partial x}+\frac{\partial p^*v/m}{\partial y}\right]+\frac{\partial p^*\sigma}{\partial\sigma},$$

和

$$\sigma=-\frac{p_{0g}}{p^*}w-\frac{m\sigma}{p^*}\frac{\partial p^*}{\partial x}u-\frac{m\sigma}{p^*}\frac{\partial p^*}{\partial y}v.$$

图14-1 气象模型中使用的一些复杂公式

这些模拟模型的计算花费是很高的。考虑到方程式的复杂性以及这些模型必须时刻保持真实性，所以只有高速的并行计算机才能在合理的时间内计算出它们。

天气预报

“早晨为红色天空，海员警告”是常被引用的天气预报。在计算机出现之前，天气预报是以民俗和观察为基础的。20世纪50年代早期，出现了第一批为天气预报开发的计算机模型。这批模型采用了非常复杂的偏微分方程。随着计算机逐渐改进，天气预报的模型也越来越复杂。

如果天气预报员使用计算机模型来预报天气，那么为什么电视和收音机预报的同一个城市的天气会不同呢？为什么它们有时是错的？计算机模型用于辅助天气预报，而不是代替预报员。计算机模型输出的是预测的将来的变量值。这些值的含义是由预报员决定的。

注意，上一段引用了多个模型。存在不同的模型，是因为它们采用的假设不同。但是，所有计算机模型都用间隔相同的网格点来模拟地球表面和表面之上的大气层。两点之间的距离决定了网格框的大小或分辨率。网格框越大，模型的分辨率越低。Nested Grid模型（NGM）的水平分辨率是80km，垂直方向有18个分层，大气层被划分成了18层方块。小方块的网格嵌套在大

方块的网格中，以便聚焦在特定的地理区域。NGM模型每隔6小时预报一次未来48小时的天气。

海啸观测

海啸专家在开发一种更好的方式，以便人们能够知道何时会有海啸。过去，海啸预警系统是以地震检波器为基础的。不过这种方法却令人头疼不已，因为并不是每一次地震都会引发海啸，这样造成的错误预警不断。现在，科学家利用放置在海底电缆上的传感器来检测海啸经过海面时引起的非常轻微的波动。当传感器检测到海啸时，放置在它附加的浮标就会通过卫星把信号发送回地面。

Model Output Statistics (MOS) 模型由一组为美国各个城市定制的统计方程构成。ETA模型是以考虑地形特征（如山脉）的ETA坐标系统命名的，它是类似于NGM模型的新模型，分辨率更高（29km）。[5]

天气模型的输出既可以是文本格式，也可以是图形格式。天气预报员的工作是解释所有输出。任何优秀的天气预报员都知道，不同模型输出结果的好坏都与微分方程的输入有关。输入数据的来源包括无线电高空测候器（测试高空的湿度、气温和气压）、无线电探空测风仪（测试高处的风速）、飞行器的观测报告、地面观测报告、卫星和其他遥感数据源。任何一个输入变量的小错误都会在积分过程中使结果中的错误不断增多。另一个问题是模型的分辨率太低，以至于天气预报员不能靠直觉精确地解释结果。

不同的天气预报员可能会相信预测的结果，也可能从其他因素判断出预测有错。此外，不同的模型还可能提供冲突的信息。哪些信息是正确的，是由天气预报员决定的。

飓风跟踪

由于飓风跟踪的模型是应用于移动目标的，所以它们叫做*浮动模型*。也就是说，模型预报的飓风的地理位置是变化的。Geophysical and Fluid Dynamics Laboratory（GFDL）为了改进飓风登陆地点的预测功能，开发了一种最新的飓风模型。

GFDL飓风模型于1995年开始运作。直到National Weather Service的高性能超级计算机用于并行计算之后，该模型中的公式计算速度才能够满足预报的需要，并行计算的运行时间比顺序计算的时间少18%。图14-2展示了这种模型在飓风跟踪方面比以前的模型进步了很多。

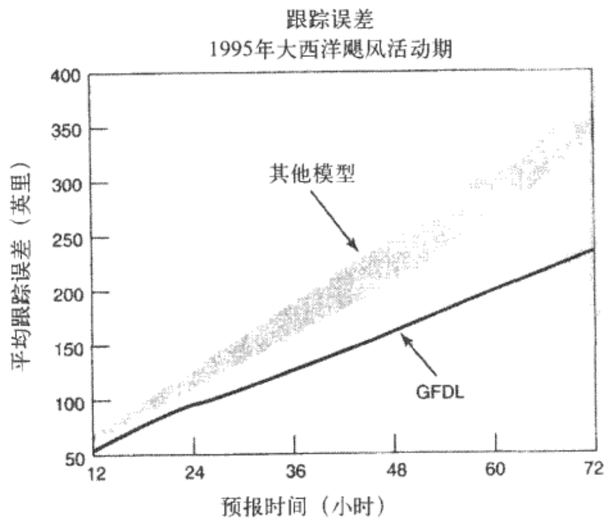


图14-2 飓风模型的改进

一些研究员正在研发把其他模型的输出组合在一起的模型。这种组合模型叫做“超级组合”，其生成的结果比独立模型生成的结果更好。与独立模型相比，该模型运行的时间越长，结果越好。在预测未来三天的飓风轨迹时，组合模型的误差为21.5mph（英里/小时），而独立模型的误差为31.3mph到32.4mph。

专用模型

气象模型可以被改编为专用模型。例如，大气运动的数字模型模拟和空气化学模型组合在一起，可以为各种空气质量应用判断大气的气流和扩散情况。其中一项研究分析了美国亚利桑那州的大峡谷地区的地形在空气污染的移动中所起的作用。

另一项研究表明，在模型演化过程中，通过消化和吸收观察到的数据，模型的性能可以比采用最初观察到的数据有极大的提高。这点考虑到了专用模型采用的输入是改进了的大气数字表示法。[6]

高级气象建模系统可以用于为军事或航空业的复杂系统提供指导。例如，天气对炮弹运动有影响，在战场上这是必须考虑的因素。在航空业中，气象数据有多种用途，包括决定要携带多少燃料以及决定何时移动飞机以避免雹灾等。

14.1.6 其他模型

就某种意义来说，每个计算机程序都是一种模拟，因为程序表示的是在问题求解阶段设计的解决方案的模型。当程序执行时，就模拟了这个模型。但我们并不想深入探讨，否则这一节将无休无止了。不过，有几个明显的领域使用了模拟。

股票市场是否将继续走高？零售价格是否会上扬？如果增加广告投入，销售额会增加吗？预报模型可以帮助解答这些问题。不过，这些预报模型不同于天气预报的模型。天气模型的基础因素是大家所熟知的，可以用流体力学和热力学的偏微分方程建模。商业和经济模型则是基于变量的历史数据建模的，因此它们采用回归分析作为预测的基础。

地震模型能够预测地震波在地壳中的传播。地震波既可以来自自然事件（如地震或火山爆发），也可以来自人为事件（如受控制的爆炸、贮水引起的地震或（工业或交通造成的）文明噪音）。对于自然事件，传感器会检测到地震波，建模，用观察到的数据作为输入，由此能够决定引起波动的原因和震级。对于人为事件，根据事件的规模和传感器数据，整个模型可以映射地球的表面。这种模型用于探测石油和天然气。地震数据用于在开始钻探前，给地质学者提供油藏和气藏高度详细的三维地图，从而最小化钻探干井的可能性。

触觉交流

触觉技术通过触觉与用户进行交互。它模拟与感觉（如气压、温度和纹理等）相关的触觉。强力反馈方向盘和操纵杆就是这种简单的触觉装置的实例。更复杂的实例包括把用户的手指放入绑在由计算机控制的机器人胳膊上的橡胶杯子中，以造成与硬表面接触的假象，此外还包括模拟能被触觉感知的三维对象。这些设备被用作外科手术模拟工具来训练医生。在出现这样的系统之前，实习外科医生只能在桔子上进行练习。

14.1.7 必要的计算能力

我们介绍的连续模型中的许多公式都是多年前就已经开发好的。也就是说，定义模型中的实体之间的相互作用的偏微分方程是众所周知的。但是，基于这些公式的模型并不能及时

地模拟出来以备使用。20世纪90年代中期引入的并行高性能计算改变了这一格局。更新、更大、更快的计算机使科学家们能够在更短的时间内解决更大范围中更复杂的数学系统。这些新机器的运行速度足以解决复杂的方程，及时地提供答案。数字天气预报与其他应用不同的是必须赶超时间，昨天的天气预报如果今天还没有收到，那就没有任何用途了。

14.2 计算机图形学

我们大体上可以将计算机图形描述为计算机屏幕上的像素值的设置。记得第3章讨论过的计算机图像吗？图像是由红、绿和蓝值定义的一组像素值。虽然那时的讨论指的是能够扫描并显示在计算机上的图片，但它适用于所有显示在计算机屏幕上的东西。

计算机图形学在计算机科学的许多领域都扮演着一定的角色。最常见的应用是现代操作系统中的图形用户界面（GUI）。文件和文件夹都由屏幕上的图标表示，图标还能说明文件的类型。与计算机的交互包括指向、点击和拖曳，这些都会改变屏幕上显示的图形。

计算机图形学决定了如何设置像素的颜色来显示图标以及在屏幕上拖曳图标时如何改变像素值。

字处理软件和桌面出版软件也是计算机图形学的应用。通过设置像素在屏幕上的值，它们能够展示打印出的文档是什么样的。也许在你想到计算机图形学时不会考虑屏幕上的黑白文本，但它们确实也是其中的一部分。用户手册中的插图也是由计算机图形学生成的。这个应用在生成图像时采用了一些特殊技术，会高亮显示正在处理的部分，而不是创建出完整的图像。

公司也利用计算机图形学来设计和制造产品。工程师使用计算机辅助设计（CAD）系统，采用几何建模技术创建新元件的规范，如图14-3所示。这些组件可以显示在屏幕上，甚至可以对某些可能断裂的点进行压力测试。这些绘图最终会用于指导装配线生产这些组件。

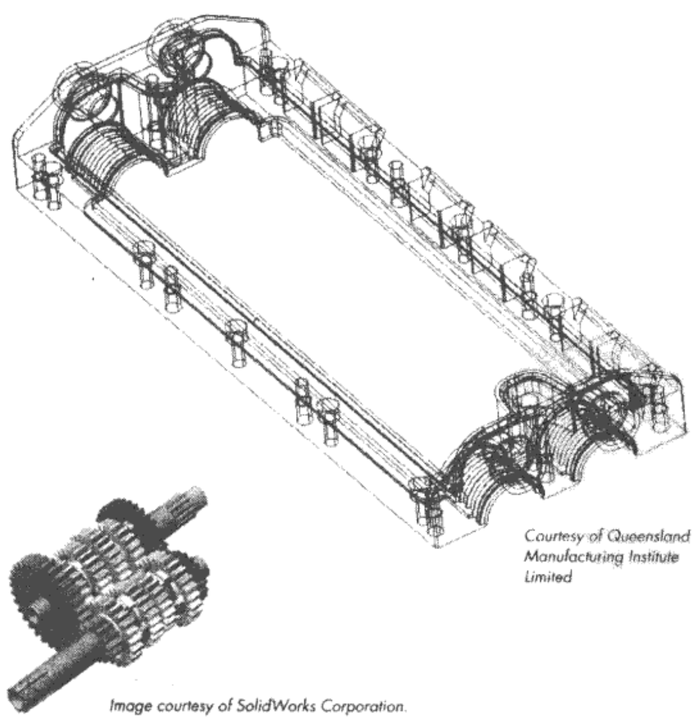


图14-3 几何建模技术

艺术家以多种方式应用计算机图形学。有些艺术家把计算机当作高科技的画布。有了绘图程序，艺术家就能用计算机代替画笔和画布进行创作。利用图像处理软件，摄影家能够润色照片，或者通过合成多个图像制作出特殊效果。此外，艺术家还用计算机作为艺术创作的主要部分。例如，1982年Jane Veeder创建了WARPITOUT计算机装置，用户可以用它拍摄数字照片，然后在它成为近代画的轮换陈列室的一部分之前，处理这些照片。

毫无疑问，科学实验和模拟会生成大量的数据。只在纸上研究数字的科学家可能会错过这些数据中的趋势或模式。一种分析方法是用图形表示数据的科学可视化。有了科学可视化系统，用户就能够修改不同值的颜色，通过数据建立交叉的部分，以便协助发现模式或趋势。一个相关的应用是医学成像。诸如CT、超声波和核磁共振这样的测试结果都以图形显示，医生或者技术人员能够利用它们进行诊断。

虽然存在大量的计算机图形学的应用，但是当你想到计算机图形学时，很可能想到的还是计算机游戏、动画片或者电视和电影中的特效。这些都是计算机图形学最有趣的应用，也是最复杂的应用。这种复杂性源自于需要模拟非常复杂的过程，即光与对象的交互、简单和复杂对象的形状建模以及人物和对象的自然移动。本节后面的部分将更详细地介绍这些主题。你会看到，计算机图形学中有很多细节，这使它成为一门既复杂又有趣的学科。由于计算机图形学可以用整本教科书来讲解，所以这一节只是给出一些基本提示。

14.2.1 光的工作原理

人的视觉系统能够发挥作用是因为物体可以反射光线，使光线进入我们的眼睛。眼睛的晶状体在光线触及眼底时会把它集中起来。眼底由视锥和杆状状胞构成，它们会对投射而来的光线产生反应。根据光线的波长，视锥可以分为长、中和短三种。长视锥反应的是红色，中视锥反应的是绿色，短视锥反应的是蓝色。杆状状胞只对光的强度有反应，所以它们缺乏颜色敏感度。人的视觉系统和大脑可以解释视锥和杆状状胞的反应，从而确保我们能看到面前的物体。

现实世界的光线会投射物体，然后反射回来。虽然我们认为只有镜子和抛光的物体才能反射光线，但事实上所有物体都可以反射光线。反射的光线量由可用的光线量决定。晴天看到的物体就比阴天或者夜晚看到的物体清楚。

除了光线量之外，物体的外观还受物体成分的影响。例如，塑料、木头和金属的属性不同，用它们制造的物体看起来就不同。塑料物体中嵌有颜色粒子，但表面非常光滑。无论物体本身是什么颜色，在高光下看来都与光的颜色一样。木制物体受木头中的纹理影响，这些纹理会对光进行漫反射。在显微镜下可以看到金属物体的表面凹凸不平，所以它们的表面发亮，但不像塑料物体那么亮。

想象一面平面镜。镜子所指的方向可以用与镜子表面垂直的法向量（N）来说明，如图14-4所示。光线在镜子上的反射角（θ）与光线进入的方向和法向量之间的夹角相同。如果你位于视向量（V）的方向，那么你所看到的会受所有这些向量的影响。整个过程非常复杂，因为光线可能从不同的方向投射到镜子上。当你从镜子中看到自己时，光线在进入你的眼睛之前，已经从各个方向经过了脸和衣服的反射。

阴影也是现实世界的一个重要组成元素。它给我们提供了物体和光源位置的视觉线索。此外，它还能给我们提供两个物体相对位置的线索。例如，如果两个物体在一起，那么一个

物体投射的阴影就会与另一个物体投射的阴影非常接近。随着两个物体渐渐离开，阴影也会根据光线条件发生变化，甚至会消失。这就解释了为什么早期手绘的卡通看起来很奇怪，因为其中有些人物有阴影，有些却没有。Micky Mouse有阴影，但是Fred Flintstone却没有。结果就是Mickey看起来是在地上走，而Fred看起来则像在空中飘。

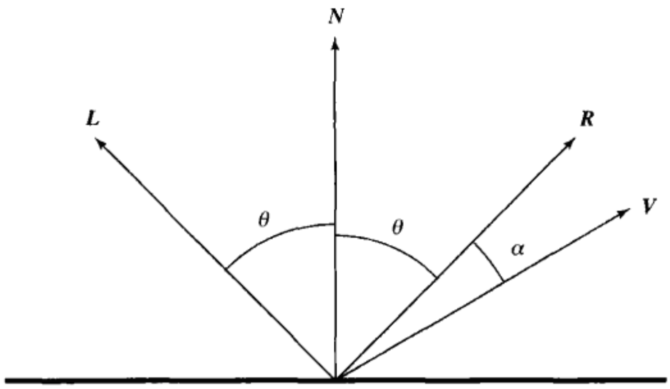


图14-4　法向量（***N***）、光线向量（***L***）、视向量（***V***）和反射向量（***R***）

要生成真实的图像，计算机必须进行计算，以模拟光和物体之间的交互、各种纹理的物体的不规则表面以及光线强度随位置在阴影中的变化。这些计算可能花费大量的时间。动画片和电影特效的效果看起来比计算机游戏的好，就是因为游戏中用到了一些简化的算法来实时地生成图像。这种处理的另一个重要元素是如何在图像中表示物体的形状。接下来将讨论这一点。

14.2.2　物体形状

物体的形状也会影响物体的外观。如果一个物体是平的（如镜子），那么物体上的任何位置都不存在法向量。如果物体不是平的，那么各个位置的法向量的方向都不同。这种法向量法向的变化改变了突出的形状，这就给了我们关于物体形状的视觉线索。

还记得数学课上用方程描述线、面、球体、圆柱体和其他形状的物体吗？计算机图形学使用这些方程说明物体的形状。环顾一下四周，你会发现物体的形状各异，比那些简单的数学对象复杂多了。计算机图形学还提供了描述曲面的数学方式，这样就可以把复杂的物体定义为独立曲面的集合。

即使真实世界中的物体是实心的，计算机图形学也只处理物体的表面，因为我们看到的只是表面。此外，这些数学方程式只能定义平滑的表面，而真实的物体表面是不规则的。例如，砖块和混凝土块的表面是粗糙的，与光滑的表面相比，这种粗糙的表面会向各个方向散射光线。图形软件利用纹理映射技术模拟粗糙的表面。

14.2.3　光模拟

在图形学中，许多技术用于模拟光和物体之间的交互。有些技术比较简单，而有些技术则计算起来非常复杂。一般来说，对光在物体上的一点的交互的模拟叫做照明模型，而利用照明模型来确定整个物体的外观的处理叫做明暗处理模型或者明暗处理。创建整个图像的过程叫做绘制。

最早的照明模型出现在1971年，它使用了三种元素，即环境光、漫反射和镜面反射。环境光是没有方向的通用光。有了这种光，我们才能看到没有光指向的物体。当光直接投射到物体表面时就会发生漫反射。这种反射发生在各个方向，是由入射方向和法向量之间的夹角决定的（即图14-4中的θ）。入射角越小，漫反射的影响越大。镜面亮点是由于镜面反射而出现在物体上的亮点。镜面反射是由反射方向和观察者方向之间的夹角决定的（即图14-4中的α）。这个角度越小，镜面反射的影响越大。物体的外观是由环境光、漫反射和镜面反射共同决定的。虽然这种模型很早就被研发出来了，但它依然是当今图形软件中常用的照明模型。

这种照明模型有一个著名的问题，即它使得所有物体看来都像用塑料制成的。因此，对它生成的金属物体或其他纹理的物体，都要进行一些调整。此外，照明模型也不能处理透明物体或者具有镜子那样的表面的物体。

第二种明暗处理方法叫做光线跟踪。采用这种方法，要将观察者的位置标识为空间中的一点。然后判断显示器（即要绘制图像的地方）的位置。现在，可以从观察者的位置开始绘制图像中的一条线的每个像素。这些像素有下面几种情况：如果这条线没有遇到任何物体，该像素将被设为背景色；如果它遇到了物体，就会执行照明计算，该像素将被设为计算出的颜色。如果遇到的物体具有反射性，如镜子，就会计算反射线的方向，然后计算该方向上的像素的颜色。如果遇到的物体是透明的，那么计算折射线的方向，然后计算该方向上的像素的颜色。更复杂的物体可能既有反射性又是透明的，那么会执行两种计算，结果会被组合在一起。因为光线传播的可能性就这么几种，所以光线跟踪既可以处理透明物体，也可以处理有反射性的物体。

你可能注意到了，有时你的T恤衫的颜色会反射到你的脸或胳膊上。这种现象叫做颜色扩散。另一个例子是当某人穿了一件亮红色的T恤衫站在白色的墙边时，这个人旁边的墙看起来会是粉色的，因为光线在投射到墙上之前经过了红色T恤衫的反射。迄今为止讨论的明暗处理方法都不能模拟这种光交互，不过一种叫做辐射度算法的技术可以处理颜色扩散。在辐射度算法中，光线被当作能量。这种复杂的算法计算一个场景中有多少能量从一个物体传递到另一个物体。对于一个大物体（如墙）来说，各部分接收到的能量不同，所以在进行能量计算之前，会把大物体分割成多个小物体。

在一个场景中，两个物体之间传递的能量的量是由两个物体相距多远以及物体的指向决定的。两个物体相距越远，传递的能量越少。两个物体相距越近，传递的能量越多。这一过程会更加复杂，因为物体A会把能量传递给物体B，相反，物体B也会把能量传递给物体A。此外，物体A能够传递给物体B的能量还部分地由物体A从物体B得到的能量决定。同样，物体B传递给物体A的能量也由物体A传递给物体B的能量决定。

辐射度算法极其复杂，不仅因为要考虑所有潜在的能量传递组合，还因为一个场景中可能存在100 000个需要考虑能量传递的物体。

14.2.4 复杂对象的建模

前面介绍过简单物体的形状可以用简单的数学对象和曲面建模。在现实世界中，许多物体的形状及其与光线的交互方式要复杂得多。这是图形学研究人员正在研究的一个领域，即如何在合理的时间内生成一个自然现象的真实模拟。这一节将大概看一下其中涉及的问题。

自然景观要求有看起来真实的地形，看起来合理的溪流，看起来自然的植物，这些是对图形学的综合挑战。图14-5展示了一个计算机生成的自然景观。可以用不规则碎片模型或腐蚀模型对地形建模。*不规则碎片模型*采用的是中点细分技术，即从1个三角形碎片入手，找出每个边的中点，连接这些顶点形成新的边，就构成了4个三角形碎片。对这4个三角形碎片分别重复上述过程，就生成了16个三角形碎片。这个结果自身并不那么有趣。但是，如果在细分过程中把中点上下移动一些，那么就能生成不规则的地形（如图14-6所示）。*腐蚀模型*可以用于构造溪流和它周围的地形。在腐蚀模型中，首先选定溪流的起点和终点，然后让溪流沿着地形随机流动。溪流的每个位置都设置了地形高度，这样溪流周围的区域就有不规则的高度。



图14-5　计算机生成的自然景观

植物生长建模采用了语法和可能性方法。基于语法的树模型在说明植物的各个部分如何变化时采用的规则与英语语法相似。例如，一条规则可能说明嫩芽变成了花，而另一条规则则可能说明嫩芽变成了树枝，在树枝尾部还长了一个嫩芽。不同的规则集合创造出的植物不同。在一套集合中的选择不同，生成的同一类型的植物也不同。根据植物的复杂度，用5～10条规则就可以描述一种植物的生长。采用可能性模型，要研究真正的植物，看它们是如何生长的。事件的可能性有很多，例如植物的嫩芽处于休眠状态，或者长成了花然后死了，或者长成了一个新树枝或一簇树枝，或者直接死了，这些可能性都会被计量。树枝的长度和它们的相对位置也会被计量。然后计算机利用这些可能性生成植物的形状。

液体、云、烟和火对图形学来说是特殊的挑战。科学家已经开发出了方程来近似模拟液体、气体和火的表现。图形学家就利用这些方程创建这些现象的图像。在计算机图形学中为液体和气体建模时，液体或气体占用的空间被划分为立方体的单元。这些方程用到了气压、密度、重力和外部压力的数据来决定物质如何在这些单元间移动。图14-7展示了一个用这种方法生成的水的示例。基于单元的云的模型会考虑当前单元和邻近单元中的湿度和云的存在性来决定云是否应该出现在当前单元中。此外，还会用随机数来影响云的构成和移动。这些技术可以生成看似真实的云，如图14-8所示。由于烟和火是物质燃烧的结果，所以烟和火的

流动都是热度造成的。此外，还有用于火的速度和烟粒子的建模的方程，这样可以生成图14-9和图14-10所示的图像。

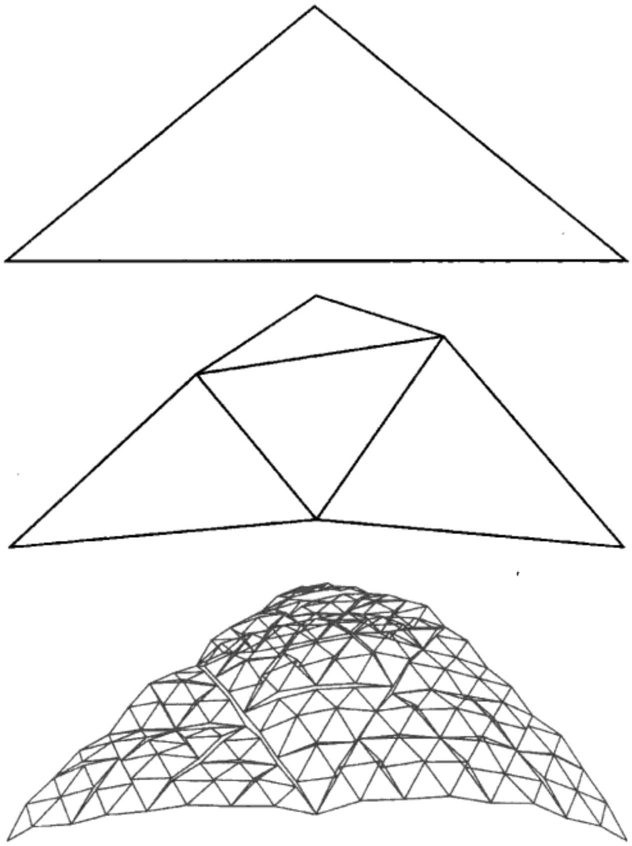


图14-6 创建不规则碎片地形的中点细分技术

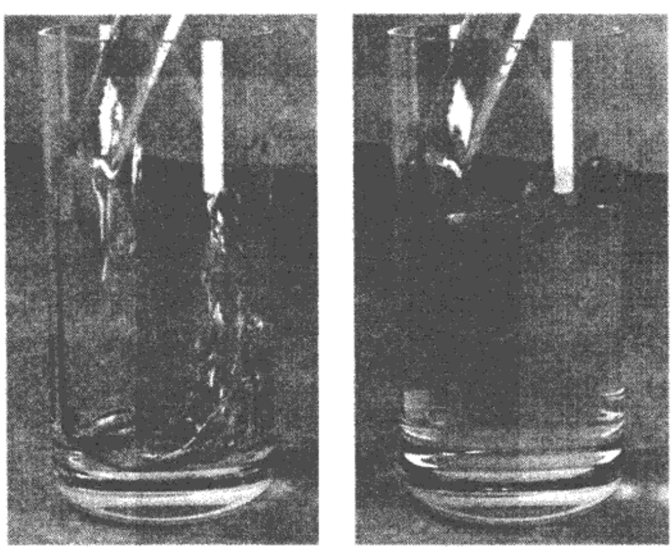


图14-7 倒入玻璃杯的水



图14-8 基于单元的云



图14-9 营火



图14-10 流动的烟

布料分为两种，即梭织的和针织的。梭织的布料由互相垂直的两套线构成。在织布过程中，水平的线上下交织在垂直的线中。加入下一条水平线时，垂直线的上下位置会改变。线的颜色和哪条线在上哪条线在下会使生成的织物呈现出不同的图案。梭织布可以拉伸，但是只能拉伸一点点，这是由采用的线和编织图案决定的。另一方面，针织布是由一根长线或纱织成的，其中用针圈打了很多结。针织布中的图案是用纱线在针圈中缠绕或回转构成的。针织布具有很强的拉伸性，而且物体周围很容易变形。

从图形学的观点来看，平铺的布并没有什么稀奇，布移动或打褶的方式才引人注目。根据构成布料的线的形状就能对梭织布的褶皱建模。由于重力作用，两个支杆之间悬挂的绳子的形状叫做*垂曲线*。对两个支杆之间悬挂的布建模可以通过使构成布的线形成垂曲线完成。这种方法的难点在于要确保布料不与自身相交，也不与其他物体相交，这个难题可以通过使用制约技术确保布料的计算不与其他物体相交来解决。当第一组计算不能阻止相交时，还可以使用其他技术把布料分离出来。图14-11展示的是一块有褶的折叠起来的布。

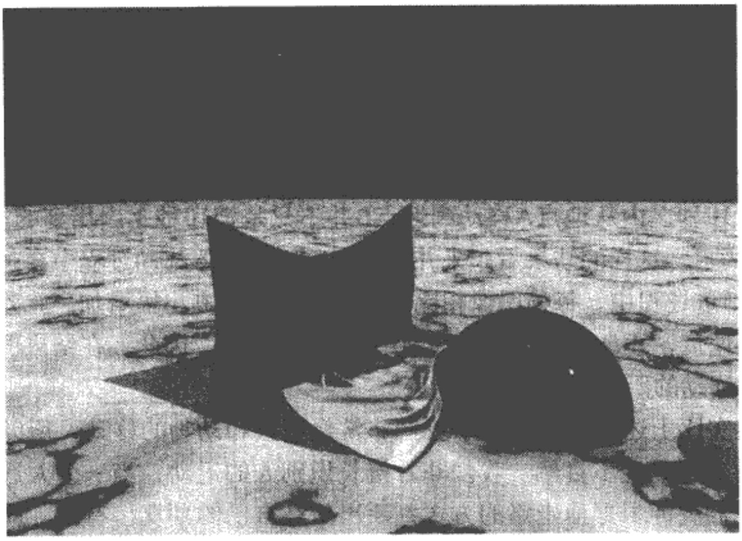


图14-11　具有折叠和褶皱的布

针织布的问题完全不同，因为当它有褶皱时，布料会被拉伸。这种拉伸会使构成布料的针圈变形。此外，由于针织布采用的是较粗的纱线，纱线会挡住部分光，所以织物中会有阴影。纱线的粗细和绒性也会影响布的外观。一种生成针织布的图形学技术是将布中的纱线的路径作为一个长曲线。曲线中的点与针织布建模平面中的点相关。要把织物放在一个物体上，这个平面就会变成物体的形状。平面的变形会改变曲线上点的位置。这些点的新位置就会反映出针织品的拉伸和折叠。绘制针织品从而变成了绘制纱线在变形曲线上的路径。

皮肤的形状和外观需要特殊的图形学技术来处理。皮肤是柔软的，它的形状取决于皮下的肌肉和骨骼。肌肉伸缩时，身体的形状会改变，所以皮肤会随之变形。此外，随着关节的移动，皮肤还会拉伸、起皱或者有折痕。在图形学中，可以用隐式表面这种高级技术对皮肤的外形建模。例如，球形方程（$x^2+y^2+z^2=r^2$）并没有明确给出球面上的x、y和z值。我们可以尝试不同的x、y和z值，直到发现一组满足r的值为止。这样，就可以隐式地找到球面上的一点。对于皮肤，有更复杂的方程用于声明隐式表面。

确定了皮肤的形状后，绘制皮肤的方式与大多数物体不同。当光打到皮肤表面时，部分光被皮肤表面的油性物质反射了，部分光则穿透了皮肤的表层。穿入皮肤的光线在从皮肤穿出之前会被下面的皮层、色素粒子和血液反射。看看你的手，你可能会注意到，除了能清楚地看到外部的皮肤外，还能看到下面的血管，甚至斑点。为了精确地绘制皮肤，图形学应用程序必须考虑这种表面之下的光线散射。在精确地绘制大理石或者牛奶这样的食品时，也要处理这种表面之下的散射。

14.2.5　让物体动起来

迄今为止，我们都是从单一图像的角度来讨论图形学的，但是游戏和动画片都要求有许多图像。一部电影每秒钟需要使用24幅图像，视频每秒钟需要30幅。这些图像要迅速显示，这样才能把图像之间的变化构成连续的动作。60分钟的动画片需要86 400幅图像，60分钟的视频则需要108 000幅图像。即使这些图像只是图像序列的一部分，绘制每幅图像的工作量也是一样的。

如果想创建出可信的物体运动，那么动画的确带来了新的挑战。如果想让运动看起来真实，就要仔细确定图像之间如何改变物体的位置。某些情况下，可以根据物理属性生成真实的运动。例如，把球抛向空中，我们可以根据万有引力定律预测球在落到地面之前是如何减速以及如何停止的。

一个物体在10秒之内从地点A移动到地点B并不只是把10秒之内用到的300幅图像等分到299个点上那么简单。这样的结果看起来并不真实，因为物体移动过程中会加速，在动画中叫做“渐进”。在渐进过程中，每帧图像相比，物体移动的距离小幅增加。此外，物体不会突然停止，而是放慢速度，逐渐停止，这在动画中叫做“渐出”。因此，在结束图像序列中，每帧图像相比，物体移动的距离小幅减少。

动画图像其实比上述更复杂。由于我们非常熟悉人和动物的移动，所以运动中即使很细小的问题都会看来明显地不自然。我们非常擅长识别人的运动，当一个人远在不能看清他的脸的地方时，我们就能根据他走路的方式判断出他是自己的朋友。有时，即使没有听到任何脚步声，我们也能判断出某人正在靠近自己。

试想一下取东西的过程。在我们要取东西时，整个手臂都会移动。如果够不到要拿的物体，肩膀就会移动，上半身会从腰部开始弯曲，或者扭腰。要做到这一点，手臂的所有部位和关节的位置都要改变。用动画模拟取东西的动作，可以先确定手需要放哪里，然后确定关节的角度。在动画过程中，可以使关节的角度从起始位置变化到终止位置。虽然这种方法能够生成所需的运动，但结果看起来可能不那么真实。如果还要计算运动的路径以避开场景中的其他物体，这个过程就更复杂了。

图形学研究者利用人类和动物运动的研究结果来开发能够自动生成更自然的运动的系统。有一种欺骗性的方式，即运动捕捉。采用运动捕捉方法，会在人身体的重要位置上放置传感器。然后人根据角色要求移动。整个移动过程中，传感器的位置会被记录下来。传感器的位置说明了角色的相应部分在移动过程中所处的位置。采用这种方法，传感器的位置就告诉了图形学应用在动画的每幅图像中角色所处的位置。这种方法非常适用于动画片，因为角色的移动是已知的。但它却不适用于计算机游戏，因为角色的移动是由游戏中所发生的情况决定的。

在日常生活中，我们做很多事情都是无意识的。我们看到物体不会想到它的光照明模式。

我们自身移动或移动物体时不会想到关节的位置，也不会想如何避免碰到其他物体。但在计算机图形学中，必须考虑所有因素，因为我们必须编写计算机程序来创建展示这种事物所需的图像。

14.3 嵌入式系统

嵌入式系统是大型系统中专用于执行有限功能的计算机。嵌入式系统通常嵌在单个微处理器芯片上，程序存储在ROM中。几乎所有具有数字界面的用具都使用了嵌入式系统，如手表、微波炉、VCR和汽车等。事实上，嵌入式系统无处不在，从电子产品到厨房用具，到汽车，到连网设备，到工业控制系统，处处都有嵌入式系统的踪影。有些嵌入式系统具有操作系统，但是大部分操作系统都是专用的，只需一个程序就可以实现全部的逻辑。[7]

早期的嵌入式系统是独立的8位微处理器，自带操作系统。今天，从8位的控制器到32位的数字信号处理器（DSP），到64位的RISC（reduced instruction set，精简指令集）芯片，嵌入式系统的种类繁多。越来越多的嵌入式系统开始采用分布式微处理器网络，这种网络可以通过有线或无线的方式通信，由常规的网络管理通信协议远程监管和控制。

事实上，术语*嵌入式系统*表达的不够清楚，因为它几乎包括除台式PC之外的所有机器。这个术语的由来是因为第一台这样的计算机被物理性地嵌入了产品或设备，不能访问。现在，这个术语指的是预编程序来执行大型系统中专门的或有限功能的计算机。其中暗示了终端用户或管理员（如果存在的话）会极少干涉这种系统。

由于一般人只是在厨房、客厅或汽车中遇到嵌入式系统，所以我们视嵌入式系统等同于硬件。但是，为嵌入式系统编写程序是必不可少的，而且要把程序烧录到系统附带的ROM（只读内存）中，以便它能实现自己的使命。由于不能在嵌入式处理器上开发和测试程序，那么如何实现它们呢？程序是先在PC上编写，然后为目标系统进行编译，生成嵌入式系统的处理器能够执行的代码。

在早期的嵌入式系统中，代码的长度和执行速度都非常重要。由于汇编语言能够最有效地简化代码，加速它们的执行，所以汇编语言一直是嵌入式系统的专用语言。甚至当C语言盛行，出现了跨平台的C编译器后，许多程序员仍然继续使用汇编语言。C程序比汇编程序大约长和慢25%，但是比较容易编写。即使是今天，ROM的大小仍然要求代码越短越好，所以汇编语言程序仍在使用中。[8]

14.4 电子商务

随着计算机应用领域的不断扩展，**电子商务**，（即通过万维网处理交易）应运而生。它包括产品和服务的营销及买卖的各个方面。近来，越来越多的人都转向以网络作为采购之先。

电子商务（electronic commerce）：使用万维网进行商品和服务买卖的过程。

从1994年网络突然进入人们的视野开始，许多人都预言它将对我们的交易方式产生巨大的影响。事实上，电子商务经过几年的时间发展成足够可信的、实用的技术，并且深入到我们的生活中。2001年“dot-com”的失败并没有削弱电子商务的力量，而是为具有合理商业模式的公司扫清了道路，使他们在网络上拥有一席之地。在这一时期，不仅出现了新的、纯粹

的在线商务，而且传统商务也开发了影响深远的在线形式。

最老的电子商务站点之一Amazon.com许多年都没有赢利，但在坚持不懈的（有时是痛苦的）发展后，它成长为杰出的电子商务站点。eBay是个流行的拍卖站点，通过它，即使没有实体商务，任何人也可以在线销售产品。目前，许多零售商都纯粹地通过eBay环境进行交易。像PayPal这样的公司，通过抽象买家的财政细节，简化了在线购买的流程，这也是电子商务成功的关键。事实上，eBay于2002年购买了PayPal。像许多在线站点一样，eBay只用PayPal作为电子支付系统。

网络技术的发展是电子商务成功的主要因素，也是隐藏在其背后的主动力。在线应用能够提供强大的用户交互功能，这对它的发展至关重要，同样，安全协议的发展以及其他允许安全转移电子基金的因素也很关键。

电子购物车是电子商务流程中的一个关键部分，有了它，用户就可以维护正在采购的物品，在一笔交易中进行结算。许多电子商务站点都跟踪用户购买的物品，并向用户提供一些他可能感兴趣的物品的建议。这是传统的店铺采购所不能比拟的。

电子商务成功的另一个重要方面在于卖家越来越了解买家是如何购物的。也就是说，现在最好的电子商务站点都为用户提供了工具，允许用户以不同的方式检索和对比自己想买的商品。同样，这种功能也是传统的店铺采购做不到的。

电子商务仍然面临的最大挑战之一是需要确保金融交易的安全性。许多人对在线交易仍持怀疑态度，不过对在线交易的信任度也在快速增长。事实上，对计算机安全性的需求比以往任何时候都更大。下一节将详细探讨安全性问题。

二手商品

有了eBay.com、craigslist.com和i-soldit.com这样的网站，人们就很容易把不想要的东西卖掉。有些专家预测，二手拍卖业最终会改变人们对所购物品的考虑。随着越来越容易卖掉不想要的东西，人们在最初购买物品时就会考虑再出售它们的价值。这种趋势会导致越来越多的商品只是被暂时拥有，人们只是租用商品，而不是买了之后丢弃掉。像Callaway Golf这样的公司已经接受了这种趋势，它允许顾客用二手物品赚取积分，然后公司再转卖这些物品。

14.5 计算机安全

下面探讨对所有计算应用都至关重要的话题——安全。我们在本书中多次讨论过安全问题。在第10章和11章中，我们探讨过与操作系统相关的安全问题，如需要确保一个用户的程序不会影响到另一个用户的程序。在第12章中，我们讨论过与信息安全相关的问题，包括密码的使用。这里，我们探讨一些其他安全问题。

当人们想到计算机安全时，首先进入脑海的通常是访问控制技术。访问控制通常会强制用户在访问程序和数据前用特定的**鉴别凭证**鉴别身份。用户的权限通常被限制在一组特定的功能上。

访问控制采用的鉴别凭证有三种。第一种也是最常用的凭证基于用户知道的信息，如用户名和口令、PIN或它们的组合。第二种凭证基于用户所拥有的物品，如具有磁条的身份卡或具有嵌入式芯片的**智能卡**。这种方法管理起来稍微复杂一些，但一般认为它比第一种方法更安全。第三种鉴别凭证以**生物特征**为基础，如指纹、视网膜模式或声音模式。这种方法实现

起来最贵，而且必须解决错误拒绝（拒绝了许可的用户）和错误接受（接受了未许可的用户）的问题。

鉴别凭证（authentication credentials）：用户访问计算机时提供的用于识别自身的信息。

智能卡（smart card）：具有嵌入式内存芯片的卡，用于鉴别用户，进行访问控制。

生物特征（biometrics）：用人的生物特征（如指纹、视网膜模式或声音模式）识别用户，从而进行访问控制。

由于用用户名和口令的鉴别方法非常常见，所以在创建口令时要遵循一定的规则：一个口令至少要有8个字符，还要包括字母（大写和小写）、数字和特殊字符。字符不能拼成词典中的一个单词，无论这个单词多么晦涩。特殊的鉴别系统可能在创建口令时会强制用户遵守这些（或者类似的）规则。

将未许可的用户拒绝在计算机系统之外是维护系统安全的基本原则，但除此之外还有其他威胁安全的方法。我们来看一些专门为攻击系统编写的程序，然后看看各种攻击计算机系统的方法。

14.5.1 恶意代码

任何明确地想绕过鉴别机制和/或执行未许可的功能的程序代码都可以定义为**恶意代码**。这种代码是通过网络或移动存储（如记忆棒和软盘）转移到计算机的。恶意代码可能会造成严重的损害（如数据毁坏），也可能只是制造一些小麻烦（如弹出讨厌的消息）。

*计算机病毒*是描述恶意代码最常用的术语，即使有时攻击造成的是另一种问题。所谓**病毒**，就是把自己嵌入另一个程序的程序。被感染的文件是病毒的宿主。执行宿主代码就会执行病毒代码。

蠕虫也像病毒一样，是一种可以自我复制的程序，但是它不需要宿主，而是作为独立的程序运行。蠕虫在它使用的网络上制造问题，把自己的副本发送到其他系统，通常会消耗带宽。相反地，病毒则是通过破坏或删除文件在特定的计算机上制造麻烦。

顾名思义，**特洛伊木马**是看来在某些方面有用，但其实在执行时会带来问题的程序。甚至这种程序在运行时，看起来它也是一种善意的资源，这使得用户很难追踪到它。与蠕虫相似，特洛伊木马是独立运行的程序；与病毒相似，它的意图是在执行它的计算机上制造麻烦。

所谓**逻辑炸弹**，是在某个特定的系统事件发生时执行的恶意代码。通常它被设置为在某个特定的日期和时间执行，不过许多类型的事件可以触发它。

恶意代码（malicious code）：一种计算机程序，尝试绕过正当的鉴别，执行未许可的功能。

病毒（virus）：能够自我复制的恶意程序，通常嵌入在其他代码中。

蠕虫（worm）：一种独立的恶意程序，目标通常是网络资源。

特洛伊木马（Trojan horse）：伪装成善意资源的恶意程序。

逻辑炸弹（logic bomb）：一种恶意程序，被设置为在某些特定系统事件发生时执行。

14.5.2 安全攻击

攻击计算机系统的方法有很多。有些攻击是试图获得不正当的访问，有些则是利用开发

缺陷，还有一些则是依赖于数字通信的弱点。下面我们来分析一下这些类型的共同特征。

前面已经讨论过选择好的口令并保护它们的重要性。有些攻击执行的是**口令猜测**，即用不同的口令反复尝试登录系统或应用。人工输入许多不同的口令是不现实的，但一个计算机程序可以通过“蛮力”的方式每秒尝试几千种可能性。这种程序通常会尝试一个在线词典中的每一个单词、这些单词的组合以及与其他字符的组合，以便看看最终是否能发现用户的口令。为了部分地解决这一问题，有些鉴别系统在用户输入口令时只允许他们失败几次，然后就终止会话。

除了猜测口令外，其他攻击可能诱使用户自愿泄露信息。**网络钓鱼**就是利用看起来像某个可信环境的官方部分的网页，但实际上这个网页是用来收集诸如用户名和口令这样的关键信息的。例如，你可能会收到一封电子邮件（假设来自eBay），向你推荐一项你可能感兴趣的业务，呈现给你一个链接。你打开的页面会要求你登录。这个页面不会让你访问你的eBay账户，而是把这些信息传输给一个恶意用户，这样他就可以对你的账户进行不正当的访问。一些诸如此类的手段非常聪明，看起来绝对是官方的。对这种联系你（而不是你主动联系对方）要求提供安全信息的情况要格外小心。

口令猜测和网络钓鱼都是黑客“欺骗”计算机系统的方法。一般说来，所谓**欺骗**，就是让一个用户伪装成另一个用户的攻击方法。

所谓**后门**，是指程序的一个问题，通过后门可以对计算机系统或程序进行特殊访问，通常是授予较高的功能访问权限。程序员在系统中明确地放入一个后门，可能是为了进行测试，也可能是为了此后能绕过系统安全机制任意妄为。无论出于哪种目的，后门都是故意嵌入程序的弱点，可能引发任何安全问题。保护系统不受后门攻击的关键是组织高质量的开发过程，由多个参与者认真审核代码，从而最小化这种弊端。

开发过程也可能造成其他安全问题。即使是无意识造成的系统缺陷也可能成为聪明的攻击者利用的弱点。用户利用这样的缺陷可以造成**缓存溢出**，这样会导致系统崩溃，也可能会使用户的权限加大，这样他们就能做自己本来不能做的事情。所谓缓存，就是一块特定大小的内存区域。如果一个程序要在缓存中存放的信息超过了缓存的容量，系统就会崩溃。这是与开发过程的质量相关的另一个问题。程序员要注意防止潜在的缓存溢出。作为用户，也要重视程序的更新。这些更新通常具有一些修复功能，可以消除潜在的安全风险，这些安全风险是在开发时最初的质量保证过程中没有发现的。

在第12章中讨论信息安全时，我们提到过另一种类型的攻击。**拒绝服务攻击**并不直接破坏数据或进行不正当的访问。相反，它会使正当的用户不能访问资源，从而使系统变得根本无用。通常DoS攻击是通过网络进行的，即让大量通信包涌入站点或其他网络资源，使它保持忙碌状态，从而不能处理其他许可的用户的请求。由于规避请求数量，它甚至会造成系统自身崩溃。

另一种网络安全问题叫做**中间人攻击**。网络通信从源移动到目的地的过程中，会经过许多地点和设备。通常，这样的通信会被正常传递，没有任何问题。当某人访问通信网络的某一点，对经过的消息进行侦听时，就是所谓的中间人攻击，这通常是借助程序实现的。它的目的是截取关键信息，如电子邮件消息中的口令。第12章讨论过的加密方法可以预防这种问题。下一章还会讨论网络安全的问题。

口令猜测（password guessing）：通过系统地尝试来判断用户口令，从而获取对计算机系统的访问的企图。

网络钓鱼（phishing）：利用网页伪装成官方系统的一部分，从而诱使用户暴露安全信息。

欺骗（spoofing）：恶意用户伪装在许可用户对计算机系统进行的攻击。

后门（back door）：程序的一个要点，知道它的人都可以利用它对计算机系统进行特殊的或未经许可的访问。

缓存溢出（buffer overflow）：计算机程序的一个缺陷，会导致系统崩溃，并让用户具有过高的访问权限。

Chatbot：为用户对话设计的程序。

拒绝服务（denial of service）：对网络资源的一种攻击，可以使许可的用户不能访问系统。

中间人（man-in-the-middle）：一种安全攻击，即通过获取公匙数据截取网络通信消息。

小结

模拟是计算的一个主要领域，它涉及为复杂系统构建计算机模型，并为观察结果而用模型进行实验。模型是真实系统的抽象，在模型中，系统被表示为一套对象或特征以及管理它们的行为的规则。

有两种主要的模拟类型——连续模拟和离散事件模拟。在连续模型中，变化是由反映对象之间的关系的偏微分方程表示的。在离散事件模拟中，行为被表示为实体、属性和事件，其中实体即对象，属性即实体的特征，事件即实体之间的相互作用。

排队系统是一种离散事件模拟，其中等待时间是要分析的因素。随机数字可以模拟事件的到达和持续，如汽车开进了银行或人们进入了超级市场。气象模型和地震模型是连续模拟的例子。

计算机图形学是结合了计算机、科学和绘画艺术的领域，令人着迷。它依赖数学方程来模拟图像中要呈现的自然现象。计算机图形学把光的交互、对象的属性（如透明度和表面材质）、对象的形状和物理属性组合在一起，生成了接近真实照片的图像。

嵌入式系统是大型系统的一部分，用于执行一个小范围内的功能。电子商务是通过Internet进行买卖服务的过程。随着电子商务越来越流行，采用的安全措施也就越严格，这样才能确保Internet上交易的完整性。

道德问题：入侵大学的计算机系统，查询录取程序中某人的录取状态

曾经有入侵计算机系统的行为是正当的吗？如某些人所指责的那样，所有的入侵行为都是有害的吗？那么如果有人未经授权访问了计算机，只是为了获取与自己有关的重要信息，而不是为了获取他人的信息或损害所访问的计算机系统，这样也是不正当的吗？

2005年3月，斯坦福大学商学院的院长宣称，在申请该学院的MBA课程的人员中，有41个人未经授权访问了斯坦福大学的录取数据库。（同样的入侵行为也在其他提供MBA课程的顶级院校发生过，如达特茅斯大学、哈佛大学和麻省理工学院。）这些申请者在申请过程中了解到了斯坦福大学和其他院校使用的Apply Youself软件的安全漏洞。虽然申请人可以未经授权访问他们的电子文件，但是他们并不能决定自己是否被MBA课程录用了。而且，这些申请人只能“入侵”浏览自己的文件，并不能访问其他申请人的信息。

由此引发的问题是这种事件是否违反了隐私权法。某些在入侵案中被抓到的申请人通过自己的律师辩解道，他们并未犯罪。他们指出，自己仅仅浏览了自己的文件，并没有访问其他申请人的信息。此外，还有些申请人辩解道，由于他们浏览的文件中的信息是他们自己的，所以他们是这些信息的合法所有人（无论它是否保存在别人的数据库中）。从这些申请人（和他们的律师）的观点来看，这种事件并没有违反任何隐私权法。

抛开是否从技术上违反了隐私权法不说，斯坦福大学的计算机系统有安全隐患是显而易见的。而且，申请人显然获得了学院数据库的未经授权的访问。无论斯坦福大学的数据库中存放的数据是否与个人相关，这些信息都是存在于一个未开放给公共访问的专有数据库中。显然，这种入侵行为违反了计算机安全法。

也许用一个类比可以帮助我们理解这起事件中的关键问题。请考虑一下Sam的例子，他知道Sally的家中有一本电话簿，其中有他的信息。他闯入Sally家，把放在厨房桌子上的电话簿的一页撕掉了一部分。当Sally回家后，看到桌上的电话簿打开了，其中一页被撕掉了一块。于是她决定报警，警察发现是Sam闯进了Sally家。Sam被捕了，被指控非法闯入。Sam辩解道，他没做任何坏事。他只是把电话簿上具有他自己的姓名、地址和电话号码的部分撕掉了。因为这些信息是关于Sam本人的，是他的个人信息，所以他是信息的所有人，他可以根据自己的意愿处置这些信息。按照Sam的观点，如果必要，那么这种权利就包括进入别人的私宅。毫无疑问，无论是Sally还是警察都不同意Sam的辩解。从他们的观点看，Sam在未经Sally许可下私闯Sally家，违反了法律，而无论他的信息是否恰好在Sally家。在这个案例中，确实违反了安全法，这与它是否违反了隐私权法无关。

练习

为练习1～8中的例子找出匹配的模拟类型。

A. 连续模拟　　　　B. 离散事件模拟

1. 天气预报。
2. 股票投资建模。
3. 地震探查。
4. 飓风跟踪。
5. 预计新银行需要的出纳员数量。
6. 确定医生办公室需要的候诊室数量。
7. 天然气探查。
8. 空气化学物质传播。

判断练习9～32中的陈述的对错：

A. 对　　　　B. 错

9. 简单系统最适合模拟。
10. 复杂系统是动态的、交互式的、结构复杂的系统。
11. 模型是真实系统的抽象。
12. 模型的表示法可以是具体的，也可以是抽象的。
13. 在计算机模拟中，模型是具体的。
14. 模型表示的特征或特性越多越好。
15. 连续模拟由实体、属性和事件表示。
16. 离散事件模拟由偏微分方程表示。
17. CAD是计算机辅助制图（computer-aided drafting）的缩写。
18. 时间驱动的模拟可以看作是为每个时间值执行一套规则的大循环。
19. 用计算机程序实现的模型是抽象模型。
20. 具体模型可以用计算机程序实现。
21. 如果光源是红色，那么在绿色塑料球表面，红色是一个高亮点。
22. 计算机图形学中常用的照明模型是20世纪70年代创建的。
23. 在计算机图形学中，环境光、漫反射和镜面反射是常用的阴影模型的三个元素。
24. 计算机图形学依赖其他科学领域对图像创建所用的方程的研究。
25. 所谓电子商务，就是在线保存财务记录（如应付款）的流程。
26. dot-com的失败促进了电子商务的发展。

27. 用生物特征收集鉴别凭证是最昂贵的方式之一。
28. 计算机病毒可以通过把自身嵌入一个程序的方法来感染该程序。
29. 可以设置在某个特定系统事件（如特定的日期和时间）发生时引爆逻辑炸弹。
30. 网络钓鱼是口令猜测的一种形式。
31. 后门威胁是由受攻击的系统的程序员实现的。
32. 拒绝服务攻击不会直接破坏数据。

练习33～67是问答题或简答题。

33. 定义模拟，并给出日常生活中5个模拟的例子。
34. 构建模型的要素是什么？
35. 列举两种模拟类型，说明两者的不同。
36. 构造一个好模型的关键是什么？
37. 在离散事件模拟中，什么定义了实体之间的相互作用？
38. 面向对象的设计和模型构造之间有什么关系？
39. 定义排队系统的目的。
40. 构建排队系统的四条必要信息是什么？
41. 随机数发生器在排队模拟中扮演什么角色？
42. 一个加油站只有一台加油泵，每隔3分钟到达一辆汽车，服务时间是4分钟，为这个排队模拟编写规则。
43. 你认为练习42中的加油站能长期维持下去吗？请解释你的答案。
44. 重写练习42中的模拟，每隔2分钟到达一辆汽车，服务时间是2分钟。
45. 为航空公司预订柜台的排队系统编写规则。柜台前只有一个队列，有两名办事员，每隔3分钟到达一个客户，处理时间为3分钟。
46. 请区分FIFO队列和优先队列。
47. SIMULA对面向对象的程序设计方法有哪些贡献？
48. 气象模型一般是基于哪些域的时间方程的？
49. 气象学家需要掌握多少数学知识？
50. 为什么天气预报模型不止一种？
51. 为什么不同的气象学家使用同样的模型也会给出不同的天气预报？
52. 什么是专用的气象模型，它们如何使用？
53. 地震模型有什么用途？
54. 请区分嵌入式系统和常规计算系统。
55. 嵌入式系统的程序员是汇编语言程序设计的最后坚持者，请解释为什么。
56. 随机数发生器可以用于改变服务时间和到达时间。例如，假设20%的顾客需要花费8分钟服务时间，80%的顾客需要3分钟。如何用随机数发生器反映这种分布状况？
57. 为什么我们说模拟给出的不是答案？
58. 模拟和电子制表软件有哪些共同之处？
59. 请解释为什么阴影在图形学应用中很重要。
60. 要创建一个桌面的模型，需要使用哪种类型的数学对象？
61. 请解释为什么在计算机动画中让对象移动很困难。
62. 哪些基于网络的技术使电子商务变成了切实可行的技术？
63. 呈现鉴别凭证的三种常用方法是什么？
64. 请描述一个特洛伊木马程序是如何攻击计算机系统的。
65. 除了本章中介绍的网络钓鱼场景外，请再描述一个假定的场景。
66. 缓存溢出为什么会让计算机系统易受攻击？
67. 中间人攻击是如何实现的？

思考题

1. 优先队列（PQ）是非常有趣的结构，可以用它们模拟栈。如何用PQ模拟栈？
2. 优先队列还可以用于模拟FIFO队列。如何用PQ模拟FIFO队列？
3. 第9章介绍过图这种数据结构。图的深度优先遍历要使用栈，广度优先遍历要使用FIFO队列。你能解释为什么吗？
4. 这一章介绍的排队系统是每个服务器有一个队列。还有其他类型的排队系统。例如，在机场，通常多个服务器对应一个队列。当一个服务器空闲时，队列前端的客户将访问该服务器。你可以用模拟表示这种类型的系统吗？
5. 用优先队列还可以对哪些生活场景建模？
6. 浏览你的厨房，列出具有嵌入式系统的物品的

数量。

7. 现在CAD系统已经普及了。去一家计算机商店看看有多少能帮助你进行设计（从厨房设计到吉他作曲设计）的程序。
8. “黑客”这个词的用法曾经与现在的含义相反，它指那些熬夜到天亮埋头于编码的程序员。非常复杂的程序几乎都是黑客通宵达旦编写出来的。现在这个词则是指那些具有恶意意图的程序员。那么这个称呼对你来说是什么含义呢？
9. 如果你有机会，会不会侵入一个网站，查看自己的信息呢？
10. 关于道德问题一文中提到的Sam的辩解，你有何看法？

第七部分 通 信 层

第15章 网 络

多年以来，计算机除了在计算领域扮演着重要的角色外，在通信领域有着同样的地位。这种通信是通过计算机网络实现的。就像复杂的高速公路系统，用各种方式把公路连接在一起，从而使汽车能够从出发点开到目的地，计算机网络也构成了一种基础设施，使数据能够从源计算机传送到目标计算机。接收数据的计算机可能近在咫尺，也可能远在天涯。这一章将探讨计算机网络的一些细节。

目标

学完本章之后，你应该能够：

- 描述与计算机网络相关的核心问题。
- 列出各种类型的网络和它们的特征。
- 解释局域网的各种拓扑。
- 解释为什么最好用开放式系统实现网络技术。
- 比较家庭Internet连接的各种技术。
- 解释包交换。
- 说明各种网络协议的基本职责。
- 解释防火墙的功能。
- 比较网络的主机名和IP地址。
- 解释域名系统。

15.1 连网

计算机网络是为了通信和共享资源而以各种方式连在一起的一组计算设备。电子邮件、即时消息和网页都依赖于底层计算机网络中发生的通信。我们使用网络共享那些无形的资源（如文件）和有形的资源（如打印机）。

计算机之间的连接通常是靠物理电线或电缆实现的。但是，有些连接使用无线电波或红外信号传导数据，这种连接是**无线**的。网络不是由物理连接定义的，而是由通信能力定义的。

计算机网络中的设备不只是计算机。例如，打印机可以直接连入网络，以便网络中的每个用户都可以使用它。此外，网络还包括各种处理网络信息传输的设备。我们用通用的术语**节点**或**主机**来引用网络中的所有设备。

计算机网络的一个关键问题是**数据传输率**，即数据从网络中的一个地点传输到另一个地点的速率。我们对网络的要求一直在提高，因为我们要靠网络来传递更多更复杂（更大）的

数据。多媒体成分（如音频和视频）是使通信量大增的主要贡献者。有时，数据传输率又叫做网络的**带宽**。在第3章对数据压缩的讨论中介绍过带宽。

> **计算机网络**（computer network）：为了通信和共享资源而连接在一起的一组计算设备。
>
> **无线连接**（wireless）：没有物理电线的网络连接。
>
> **节点（或主机）**（node (or host)）：网络中任何可寻址的设备。
>
> **数据传输率（也叫带宽）**（data transfer rate (also bandwidth)）：数据从网络中的一个地点传输到另一个地点的速率。

计算机网络的另一个关键问题是使用的**协议**。我们在本书其他地方提到过，协议是说明了两个事物如何交互的一组规则。在连网过程中，我们使用明确的协议来说明如何格式化和处理要传输的数据。

计算机网络开创了一个新的计算领域——**客户/服务器模型**。计算机不再只是具有你面前的那部机器的功能。软件系统分布在整个网络中，在这个网络中，客户将向服务器请求信息或操作，服务器则对之做出响应，如图15-1所示。

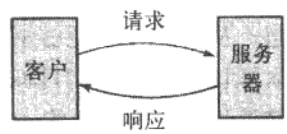


图15-1 客户/服务器的交互

例如，**文件服务器**是网络中为多个用户存储和管理文件的计算机，这样每个用户不必都有自己的文件副本。**Web服务器**是专用于响应（来自客户浏览器的）网页请求的计算机。随着我们的日常生活对网络依赖性的增加，客户/服务器关系也变得越来越复杂。因此，客户/服务器模型在计算世界中也变得越来越重要了。

此外，客户/服务器模型变得不止有基本的请求/响应功能，它开始逐渐支持并行处理，即像第4章介绍的那样把一个问题分解成若干小问题，然后用多台计算机来解决它们。使用网络和客户/服务器模型，就可以通过让客户请求多台机器执行一个问题的特定部分来实现并行处理。客户收集到每台机器的响应后再把它们构成一个完整的解决方案。

> **协议**（protocol）：定义如何在网络上格式化和处理数据的一组规则。
>
> **客户/服务器模型**（client/server model）：客户发出对服务器的请求，服务器做出响应的分布式方法。
>
> **文件服务器**（file server）：专用于为网络用户存储和管理文件的计算机。
>
> **Web服务器**（Web server）：专用于响应网页请求的计算机。

15.1.1 网络的类型

计算机网络的分类方式有多种。**局域网**（LAN）是连接较小地理范围内的少量计算机的网络。LAN通常局限在一个房间或一幢建筑中。有时它们也可能延伸到几幢相距较近的建筑。

管理LAN的各种配置叫做*拓扑*。**环形拓扑**把所有节点连接成一个封闭的环，消息在环中沿着一个方向传播。环形网络中的节点将传递消息，直到它们到达了目的地。**星形拓扑**以一个节点为中心，其他节点都连接在中心节点上，所有消息都经过中心节点发送。星形网络给中心节点赋予了巨大的负担，如果中心节点不工作了，那么整个网络的通信就瘫痪了。在**总线拓扑**中，所有节点都连接在一根通信线上，消息可以在通信线中双向传播。总线上的所有节点将检查总线传输的每个消息，不过如果消息所寻的地址不是该节点，它会忽略这条消息。图15-2展示了各种拓扑。被称为**以太网**的总线技术已经成为了局域网的业界标准。

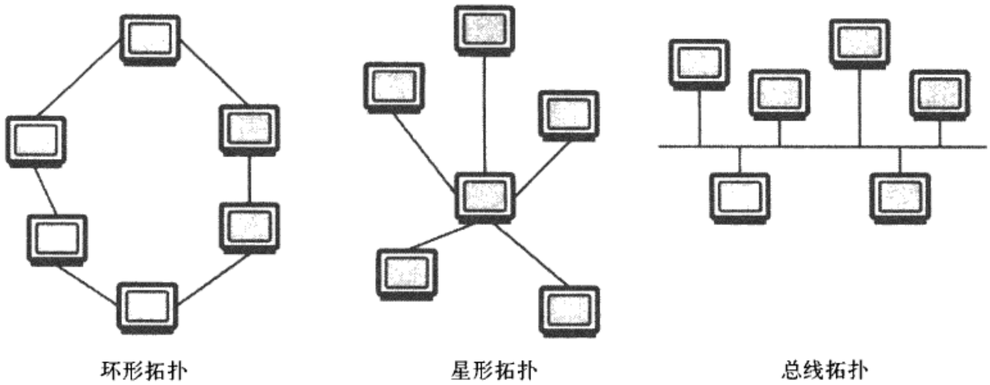


图15-2 各种网络拓扑

广域网（WAN）是连接两个或多个相距较远的局域网的网络。广域网使得较小的网络之间可以互相通信。LAN中通常会有一个特殊节点作为**网关**，处理这个LAN和其他网络之间的通信。如图15-3所示。

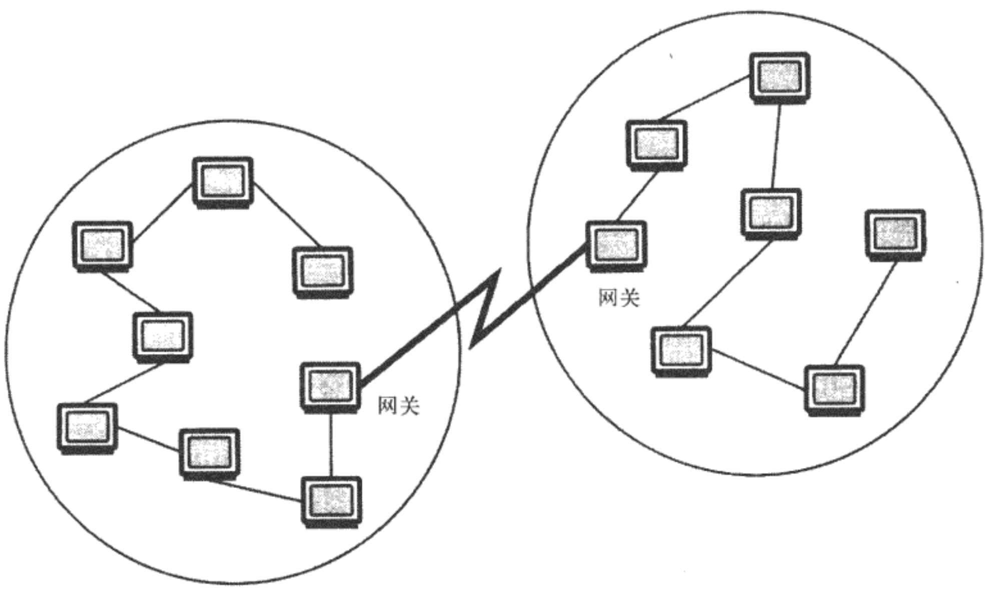


图15-3 连接两个远距离的局域网构成一个广域网

网络之间的通信叫做网际互连。我们现在所熟知的Internet本质上就是一个最大的广域网，遍布整个地球。Internet是巨大的小网络集合，这些小网络都采用相同的协议通信，而且会传递经过的消息，使它们能够到达最终目的地。

术语**城域网**（MAN）有时用来指覆盖校园或城市的大型网络。与一般广域网相比，MAN更适合于特定的组织或区域使用。为院校服务的MAN通常都与各个建筑或院系的局域网互连。有些城市在它们的地域组建了MAN，用于服务大众。城域网通常是通过无线连接或光纤连接实现的。

局域网（local-area network, LAN）：连接较小地理范围内的少量计算机的网络。

环形拓扑（ring topology）：所有节点连接成了封闭环的LAN配置。

> **星形拓扑**（star topology）：由中心节点控制所有消息传输的LAN配置。
> **总线拓扑**（bus topology）：所有节点共享一根通信线的LAN配置。
> **以太网**（Ethernet）：基于总线拓扑的局域网业界标准。
> **广域网**（wide-area network, WAN）：连接两个或多个局域网的网络。
> **网关**（gateway）：处理它的LAN和其他网络之间通信的节点。
> **Internet**：遍布地球的广域网。
> **城域网**（metropolitan-area network, MAN）：为大城市开发的网络基础设施。

15.1.2 Internet连接

那么谁拥有Internet呢？没有一个人或公司拥有Internet，甚至不能完整地控制它。作为一个广域网，它由多个小网络构成。这些小网络则通常属于某个人或某个公司。这些网络之间是如何连接的才真正定义了Internet。

Internet骨干网指的是承载Internet通信的一组高速网络。这些网络是由AT&T、Verizon和British Telecom这样的公司以及几家政府或学院支持的资源提供的。骨干网使用的都是具有高数据传输率（从每秒1.5M到用特殊光缆实现的每秒600多兆位）的连接。记住，Internet网络（包括骨干网）有大量的冗余，所以根本没有真正的中央网络。

Internet服务提供者（ISP）是给其他公司或个人提供Internet访问的公司。ISP直接连接到Internet骨干网或连接到更大的ISP。America Online和Prodigy就是两家ISP。

> **Internet骨干网**（Internet backbone）：承载Internet通信的一组高速网络。
> **Internet服务提供者**（Internet service provider, ISP）：提供Internet访问的公司。

把家用计算机连接到Internet上的方法有很多，最常用的三种是使用电话调制解调器、数字用户线路（DSL）或线缆调制解调器。下面我们将分别介绍每种连接方法。

在Internet连接的需求出现之前，电话系统早就进入了千家万户。因此，以*电话调制解调器*作为家庭网络通信的首选方式就在情理之中。术语*调制解调器*是调节器和解调器的缩写。**电话调制解调器**将把计算机信号转换成模拟音频信号，以便在电话线中传输，目的地的调制解调器将把模拟音频信号转换回计算机信号。一种音频用于表示二进制的0，另一种用于表示1。

> **电话调制解调器**（phone modem）：把计算机信号转换成模拟音频信号，然后再把模拟音频信号转换回计算机信号的设备。

> **SETI @ home**
>
> SETI@home是一个分布式的实验项目，它利用互连计算机搜寻地球外的生命（Search for Extraterrestrial Intelligence, SETI）。这是由加州大学空间科学实验室主持的一个项目。SETI@home利用用户计算机上的空闲计算资源分析Arecibo射电望远镜收集的数据，该望远镜一直在搜寻来自地球外的生命的无线电广播。到2006年年中时，该项目已经有520万来自世界各地的参与者，是吉尼斯世界纪录中有史以来最大型的计算项目。

要使用电话调制解调器，必须首先在家用计算机和永久连接到Internet的计算机之间建立

电话连接。你的ISP就是通过这个连接为你提供服务的。你每个月付给ISP一定的费用，就可以连接几台专用的计算机（最理想的是本地计算机）。一旦建立了连接，就可以通过电话线把数据发送给你的ISP，ISP将把这些数据发送到Internet骨干网。传回的数据将被路由到你的ISP，进而发送到你的家用计算机上。

因为这种方法不需要电话公司做任何特殊工作，所以实现起来非常简单。由于数据被当作语音谈话处理，所以除了在两端之外，不需要特殊的转换操作。不过这种简便是有代价的。这种方法的数据传输率被限制在模拟语音通信的数据传输率，通常最多每秒64KB。

如果把数据当作数字信号而不是模拟信号，那么电话线可以提供相当高的传输率。**数字用户线路**（DSL）就是使用常规的铜质电话线给电话公司的核心办公室传输数字数据。由于DSL和语音通信使用的频率不同，所以同一根电话线就可以满足这两种用途。

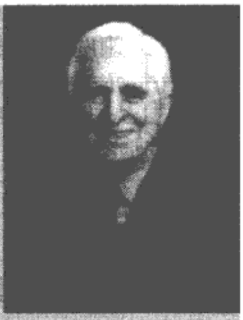
数字用户线路（digital subscriber line, DSL）：用常规电话线传输数字信号的Internet连接方式。

要建立DSL连接，你的电话公司必须是你的ISP，或者它们把电话线的使用权卖给了第三方ISP。为了提供DSL服务，电话公司必须建立专用计算机来处理数据通信。虽然并非所有电话公司都支持DSL，不过它逐渐成了一种受欢迎的连接方法。

使用DSL，不必像电话调制解调器那样用拨电话的方式建立网络连接。DSL线路在你的家用计算机和ISP的计算机之间维护了一个活动连接。不过，由于数字信号在两点间传输的过程中会减弱，所以要使用DSL技术，你家不能离电话公司的核心办公室太远。

家庭连接的第三种方式是**线缆调制解调器**。在这种方法中，传输数据的线缆就是传输有线电视信号的线缆。北美几家主要的有线电视公司都与ISP共享他们的资源，提供线缆调制解调器的服务。

线缆调制解调器（cable modem）：使用家庭的有线电视网络进行计算机网络通信的设备。



Doug Engelbart

"一支鼠标通天下。鼠标让你看尽万象，而你却忘却了其发明者的名字。"这是庆祝鼠标诞生20周年的一篇文章中的开篇语。[1]

1968年，鼠标在Fall Joint Computer会议上首次亮相，它是后来被Andy van Dam称为"The Mother of All Demos"的演示的一部分，由Doug Engelbart（就是被世界遗忘的名字）和斯坦福研究院的年轻科学家和工程师们设计。这次历史性的演示预示了人机交互和连网技术的诞生。但是，直到1981年才出现第一台使用鼠标的商业计算机。1984年，Apple Macintosh把鼠标引入了主流。到此为止，没人知道术语"鼠标"出自何处。

Engelbart是在俄勒冈州的波特兰附近的农场长大的，此时正值经济萧条时期。二战期间，他服务于Philippines的海军，担任电子技师。1948年，他在俄勒冈州立大学完成了电气工程学的学位，转移到了海湾地区。1955年，他收到了加州伯克利大学的博士学位，然后加入了斯坦福研究院。

在1962年发表的有创见性的论文"Augmenting Human Intellect: A Conceptual

Framework”中，Engelbart把计算机想象成人类交流能力的延伸和增加人类智慧的资源。他从来没忘记过自己的梦想。从此开始，他一直致力于开发使计算机和人类组织共同发展的模型，创建了他所谓的“高性能组织”。[2]

20世纪70年代和80年代中，Engelbart是Tymshare公司的高级科学家，这家公司被McDonnell-Douglas收购后，Engelbart于1989年建立了Bootstrap Institute，目的是帮助公司和组织使用他的技术。支持开源运动的程序员会协作开发高级的复杂的软件，这给了他很大的鼓舞。当前他正在设计一个可以在Internet上免费分发的开源软件系统。

虽然公众的赞誉来得有些迟，但在1987年到2001年间，Engelbart获得了32个奖项，其中包括1997年的图灵奖和2000年的美国国家技术奖。这两个奖项的获奖词如下：

（图灵奖）为了他对交互式计算的将来的创见以及为实现这种创见而发明的关键技术。

（美国国家技术奖）为他创立了个人计算的基础，包括基于阴极射线管显示器的连续实时交互技术、鼠标、超文本链接、文本编辑、在线期刊、共用屏幕的远程会议和远程协作。

DSL连接和线缆调制解调器都属于**宽带**连接，即数据传输速率至少为每秒128KB。关于DSL和线缆调制解调器哪个能统治市场的争论逐渐进入了白热化。这两种方法提供的数据传输率都在每秒1.5MB到3MB之间。

宽带（broadband）：提供的数据传输率大于128Kbps的连网技术。

DSL和线缆调制解调器的**下载**（从Internet上把数据传到家用计算机上）速度可以和**上载**（把家用计算机上的数据发送到Internet上）速度不同。家庭Internet用户的大部分数据通信都是下载网页，浏览和接收存储在网络其他地方的数据（如程序或音频和视频剪辑）。当你发了一封电子邮件，提交了一个基于Web的表单，或请求了一个新网页时，都在执行上载操作。由于下载的数据通信量远远大于上载的数据量，所以许多DSL和线缆调制解调器的提供商提供的下载速度比上载速度快。

下载（download）：在家用计算机上接收Internet上的信息。

上载（upload）：从家用计算机给Internet上的目标机器发送数据。

15.1.3 包交换

为了提高在共享线路上传输信息的有效性，消息被分割为大小固定、有编号的**包**。每个包将独立在网上传输，直到到达目的地，它们将在此被重新组合为原始的消息。这种方法叫做**包交换**。

每个消息的包可以采用不同的路由线路。因此，它们到达目的地的顺序可能与发送顺序不同。需要把包按照正确顺序排列之后再组合成原始消息。图15-4展示了这一过程。

包在到达最终目的地之前，会在各种网络的计算机之间跳跃。用于指导包在网络之间传输的设备叫做**路由器**。中间的路由器不能规划包的整个传输路线，每个路由器只知道到达它的下一个目的地的最佳步骤。最终，消息将到达一个知道目的地机器的路由器。如果由于下行机器的问题中断了路径，或者选中的路径当前具有很大的通信量，那么路由器可能会把包

发送给另一个路由器。

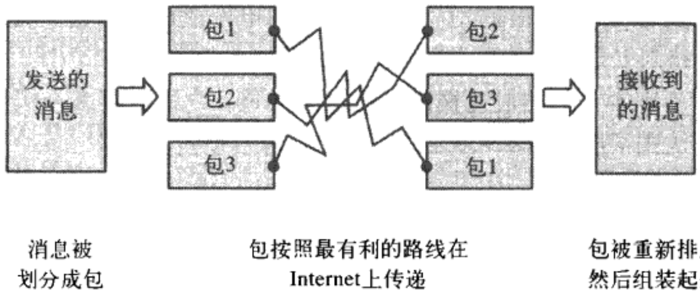


图15-4 通过包交换技术发送的消息

如果通信线跨越的距离很长（如跨海的），那么线路上将安装**中继器**，以周期性地加强和传播信号。第3章介绍过，如果数字信号减弱太多，它会损失信息。中继器会阻止这种情况发生。

> **包**（packet）：在网络上传输的数据单位。
> **包交换**（packet switching）：把包单独发送到目的地然后再组装起来的网络通信技术。
> **路由器**（router）：指导包在网络上向最终目的地传输的网络设备。
> **中继器**（repeater）：在较长的通信线路上加强和传播信号的网络设备。

15.2 开放式系统和协议

人们已经定义了很多协议来支持网络通信。由于许多原因（通常是历史原因），某些协议的地位比其他协议高。这一节将着重介绍一些Internet通信常用的协议。但在讨论具体的协议之前，需要讨论一些开放式系统的概念，以便提供一些背景。

15.2.1 开放式系统

在计算机网络发展的早期，销售商提出了许多希望商家能够采用的技术。问题是这些**专有系统**都有自己特有的差别，不同类型的网络之间不能进行通信。随着网络技术的发展，**互通性**的需求越来越明显，我们需要一种使不同销售商出售的计算系统能够通信的方式。

开放式系统的基础是网络体系结构的通用模型，它的实现采用了一系列协议。开放式系统最大化了互通性的可能。

国际标准化组织ISO建立了**开放式系统互连（OSI）参考模型**来简化网络技术的开发。它定义了一系列网络交互层。图15-5展示了OSI参考模型。

7	应用层
6	表示层
5	会话层
4	传输层
3	网络层
2	数据链路层
1	物理层

图15-5 OSI参考模型

> **专有系统**（proprietary system）：使用特定销售商的私有技术的系统。
> **互通性**（interoperability）：多台机器上的来自多个销售商的软件和硬件互相通信的能力。
> **开放式系统**（open system）：以网络体系结构的通用模型为基础并且伴有一组协议的系统。
> **开放式系统互连参考模型**（Open Systems Interconnection reference model）：为了便于建立通信标准而对网络交互进行的7层逻辑划分。

每一层处理网络通信的一个特定方面。最高层处理的是明确地与应用有关的问题。最低层处理的是与物理传输介质（如线型）相关的基础的电子或机械问题。其他层填补了其他各个方面。例如，网络层处理的是包的路由和寻址问题。

每一层的细节不在本书的讨论范围内，但是要知道，之所以存在今天我们所熟知的连网技术，都归功于开放式系统的技术和方法（如OSI参考模型）。

15.2.2 网络协议

网络协议参照OSI参考模型的基本概念也进行了分层，以便OSI参考模型中的每一层都能依靠自己的基础协议，如图15-6所示。这种分层有时叫做**协议栈**。采用分层的方法，可以在不舍弃低层基础结构的前提下，开发新的协议。此外，这样还最小化了网络协议对网络处理其他方面的影响。有时，同一层中的协议提供同样的服务，但是采用的方式却不同。

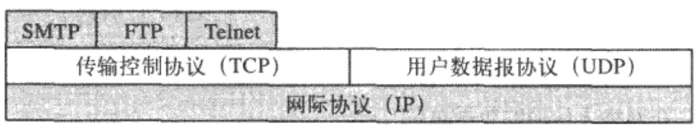


图15-6　关键网络协议的分层

> **协议栈**（protocol stack）：彼此依托的协议分层。

协议在某种意义上只是一种条约，规定了特定的数据类型必须按照特定的方式格式化。虽然文件格式的细节和数据域的大小对创建网络程序的软件开发者来说很重要，但是这里不必探讨它们的细节。这些协议的重要之处在于，它们提供了一种在连网的计算机间进行交互的标准方式。

图15-6中的最低两层构成了Internet通信的基础。其他协议有时叫做高层协议，负责处理特定类型的网络通信。这些层本质上是OSI参考模型的特定实现，以各种方式对应于该模型中的分层。让我们详细探讨一下这些分层。

15.2.3 TCP/IP

TCP是**传输控制协议**的缩写，IP是**网际协议**的缩写。TCP/IP的读法就是T-C-P-I-P，它指的是一组协议和支持低层网络通信的工具程序。TCP/IP这种写法也反映了它们之间的关系，即TCP是在IP的基础之上的。

IP软件处理的是包通过互相连接的网络传递到最终目的地的路由选择。TCP负责把消息分割成包，交给IP软件传递，目的地机器上的TCP则负责把包排序，重新组合成消息。TCP软件还要处理所有发生的错误，如一个包永远不能到达目的地。

UDP是**用户数据报协议**的缩写。它是TCP的替代品。也就是说，UDP软件的角色基本上与TCP软件一样。主要的不同之处在于TCP牺牲了一定的性能，提供了高度可靠性，而UDP更快，但不那么可靠。注意，UDP是TCP/IP协议组的一部分。由于TCP是高度可靠的，还出于一定的历史原因，所以这套协议叫做TCP/IP协议。

IP程序ping可以用于测试网络指派的可达性。每个运行IP软件的计算机都会对ping请求作出回应，这使得ping成了一种方便的测试方式，无论特定的计算机是否在运行，也无论是否

能通过网络达到它。ping是Packet InterNet Groper的正式缩写，这个名称来源于潜水艇发送一个声纳脉冲，然后侦听返回的回声所采用的术语。由于ping是在IP层运行的，所以即使高层协议没有响应，它常常也会作出反应。网络管理员之间通常把ping用作动词，如“ping一下计算机X，看它是否开着。”

另一种TCP/IP工具叫做**跟踪路由程序**，用于展示包在到达目的节点的过程中经过的路线。跟踪路由程序输出的是作为中转站的计算机的列表。

传输控制协议（Transmission Control Protocol, TCP）：把消息分割成包，在目的地把包重新组装成消息，并负责处理错误的网络协议。

网际协议（Internet Protocol, IP）：网络协议，处理包通过互相连接的网络传递到最终目的地的路由选择。

TCP/IP：一组支持低层网络通信的协议和程序。

用户数据报协议（User Datagram Protocol, UDP）：牺牲一定可靠性实现较高传输速率的网络协议，是TCP的替代者。

ping：用于测试一台特定的网络计算机是否是活动的以及是否可到达的程序。

跟踪路由程序（traceroute）：用于展示包在到达目的节点的过程中经过的路线的程序。

15.2.4 高层协议

其他协议都是在TCP/IP协议组建立的基础之上构建的。一些关键的高层协议如下：

- 简单邮件传输协议（SMTP）——用于指定电子邮件的传输方式的协议。
- 文件传输协议（FTP）——允许一台计算机上的用户把文件传到另一台机器或从另一台机器传回的协议。
- telnet——用于从远程计算机登录一个计算机系统的协议。如果你在一台特定的计算机上拥有允许telnet连接的账户，那么就可以运行采用telnet协议的程序，连接并登录到这台机器，就像你坐在这台机器面前一样。
- 超文本传输协议（HTTP）——定义WWW文档交换的协议，WWW文档通常是用超文本标示语言（HTML）写成的。第16章将详细讨论HTML语言。

这些协议都是构建在TCP之上的。还有些高层协议构建在UDP之上，主要是为了利用它提供的速度。不过，由于UDP的可靠性不如TCP，所以UDP没有TCP那么流行。

有些高层协议具有特定的端口号。**端口**是对应于特定高层协议的数字标号。服务器和路由器利用端口号控制和处理网络通信。图15-7列出了常用的协议和它们的端口。有些协议（如HTTP）具有默认的端口，但也可以使用其他端口。

协 议	端 口
Echo	7
文件传输协议（FTP）	21
Telnet	23
简单邮件传输协议（SMTP）	25
域名服务（DNS）	53
Gopher	70
Finger	79
超文本传输协议（HTTP）	80
邮局协议（POP3）	110
网络新闻传输协议（NNTP）	119
在线聊天系统（IRC）	6667

图15-7 一些协议和它们使用的端口

端口（port）：特定高层协议对应的数字标号。

Bill爵士？

在白金汉宫举行的一场私人宴会上，英国女王授予Microsoft公司创始人之一比尔·盖茨荣誉爵士称号。这一荣誉是为了表彰他在世界各地从事的慈善活动以及他为英国的高科技产业所做的贡献。

15.2.5 MIME类型

与网络协议和标准化相关的概念是文件的**MIME类型**。MIME是多用途网际邮件扩充（Multipurpose Internet Mail Extension）的缩写。虽然MIME类型没有定义网络协议，它定义了给文档（如电子邮件）附加或加入多媒体或其他特殊格式的数据的标准。

应用程序根据文档的MIME类型可以决定如何处理其中的数据。例如，用于阅读电子邮件的程序会分析电子邮件附件的MIME类型，以决定如何显示它（如果可以的话）。

MIME类型（MIME type）：定义电子邮件附件或网站文件的格式的标准。

许多常用应用程序创建的文档和来自特定领域的数据都有MIME类型。例如，化学家和化学工程师为各种与化学相关的数据类型定义了一大套MIME类型。

15.2.6 防火墙

防火墙是一台机器，它的软件将作为网络的特殊网关，保护它免受不正当的访问。防火墙将过滤到来的网络通信，尽可能地检查消息的有效性，可能会拒绝某些消息。防火墙的主要作用是保护（从某种程度上讲是隐藏）驻留在它“后边”的一组管理较松懈的机器。图15-8展示了这个过程。

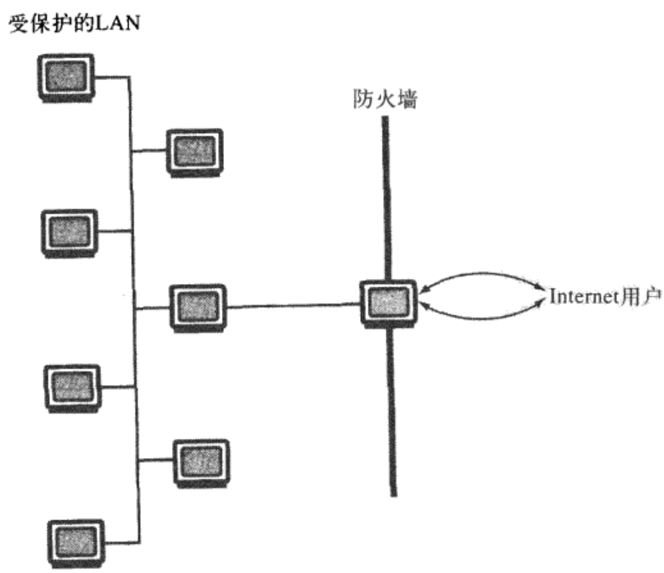


图15-8 保护LAN的防火墙

防火墙会强制执行一个组织的**访问控制策略**。例如，一个特定的组织可能只允许它的用户

和外界以电子邮件进行网络通信，拒绝其他任何通信方式，如站点访问等。另一个组织可能允许用户自由访问Internet的资源，但不想让一般的Internet用户渗透到它的系统中或访问它的数据。

> **防火墙**（firewall）：一台网关机器，它的软件通过过滤网络通信来保护网络。
>
> **访问控制策略**（access control policy）：一个组织建立的一组规则，规定了接受和拒绝什么类型的网络通信。

组织的系统管理员将为他们的LAN设置防火墙，接受"可接受"类型的通信，拒绝其他类型的通信。实现这一点的方法有很多，最简单的一种是用端口号拒绝通信。例如，可以建立防火墙，通过拒绝由端口23进入的所有通信，能够阻止LAN之外的用户创建对LAN之内机器的telnet连接。

更复杂的防火墙系统能维护有关流经它们的通信的状态的内部信息或存储数据本身。防火墙能够决定的通信状态越多，就越能够保护它的用户。当然，这种安全性是有代价的。有些防火墙会给网络通信带来明显的延迟。

15.3 网络地址

当你通过一个计算机网络进行通信时，最终都是在与世界上某处的另一台计算机通信。标识特定的机器以建立通信是一种相当复杂的机制。

主机名是Internet上的计算机的唯一标识。主机名通常是易读懂的单词，中间由点号分隔。例如：

```
matisse. csc. villanova.edu
condor. develocorp. com
```

在处理电子邮件地址和站点时，我们倾向于使用主机名，因为它们容易理解和记忆。但是，网络软件却要把主机名翻译成对应的**IP地址**，这样更便于计算机使用。IP地址通常是4个十进制数，中间由点号分隔。例如：

```
205. 39. 155.18
193. 133 20. 4
```

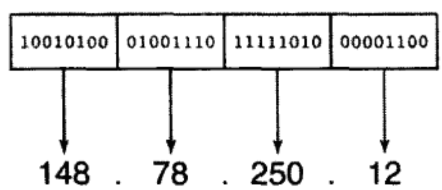


图15-9 4字节长的IP地址

一个IP地址长为32位，IP地址中的每个数对应IP地址中的一个字节。由于一个字节（8位）可以表示256种事物，所以IP地址中的数字的范围是0到255。如图15-9所示。

> **主机名**（hostname）：由点号分隔的单词组构成的名字，唯一标识了Internet上的机器；每个主机名对应一个IP地址。
>
> **IP地址**（IP address）：由点号分隔的四个数值构成的地址，唯一表示了Internet上的机器。

有种假设认为，由于主机名和IP地址都是由点号分隔成几个部分，所以它们的每个部分之间也是对应的。事实并非如此。首先，IP地址一定有4个值，而主机名中包含几部分是不确定的。

可以把IP地址分割成**网络地址**，指定一个特定的网络和**主机号**，后者指定了网络中的一台特定机器。如何分割IP地址是由它表示的网络"类"决定的。不同大小的网络具有不同的网络类（A、B和C）。

网络地址（network address）：IP地址中指定特定网络的那部分。

主机号（host number）：IP地址中指定特定网络主机的那部分。

A类网络把第一个字节作为网络地址，其他三个字节用作主机号。B类网络把前两个字节作为网络地址，后两个字节用作主机号。C类网络把前三个字节作为网络地址，最后一个字节用作主机号。

考虑一下采用这种寻址方法，每种网络的取值范围。A类网络比较少，但是每个网络中的主机较多。另一方面，C类网络很多，但每个网络中的主机很少（最大256）。大多数组织使用的是C类网络地址，A类和B类网络地址是为大型组织和ISP保留的。

整个Internet协议都是以32位的IP地址为基础的。如果Internet上的设备量继续增长，最终可用的地址空间会被耗尽。连网界一直在为如何处理这个难题争论不休。

Captcha码

Captcha码是一组扭曲的杂乱字母和数字，某些网站要求用户输入这种代码，以阻止垃圾广告使用的自动程序的访问。Captcha是Completely Automated Public Turing Test to Tell Computers and Humans Apart（分辨计算机和人为操作的全自动公共图灵测试）的缩写。视力有问题的用户一直抱怨这种代码，而且为了抵制垃圾广告，这种代码越来越长，以致激怒了很多用户。程序员正在致力开发一种新的变体，以便人们能更容易输入，让计算机程序更难解密。有些公司已经加入了Captcha码的有声版本，还有一些网站开发的Captcha可以解简单的方程或者回答简单的问题。

域名系统

主机名由计算机名加**域名**构成。例如，在主机名

```
matisse.csc.villanova.edu
```

中，matisse是计算机名，csc.villanova.edu是域名。域名由两个或多个部分组成，它们说明了计算机所属的组织或组织的一个子集。在这个例子中，matisse是Villanova大学的计算机科学系的一台计算机。

域名（domain name）：主机名中说明特定的组织或分组的部分。

域名仅限于由特定组织控制的一组特定网络。注意，两家组织中的计算机可以重名，因为从域名可以分辨出引用的是哪一台计算机。

域名中的最后一部分叫做**顶级域名**（TLD）。图15-10列出了主要的顶级域名。

有些TLD（图15-10中带星号的）从Internet建立伊始就存在了，而其他的则相对较新。

一种TLD通常用于一种特定类型的组织，如.com用于商业组织，.edu用于大学和学院。有些TLD是受严格控制的（如.edu），只有真正属于这种类型的组织才能注册。其他的TLD则没那么严格。除美国外，其他国家通常采用两位的国家代码作为顶级域名。图15-11列出了部分国家代码（国家代码有几百个）。

由于.com、.org和.net这样的域名不受控制，所以最初任何人或组织都可以注册自己的域名，只要这个名字还没有被使用即可。随着Internet不断扩大，命名系统成了问题。最令Internet新用户烦恼的是最好的域名已经被他人占用了。有时，域名是被类似的公司占用了，

但有时则是被某些人占用了，他们尽可能多地申请常用的名字，希望能卖给大型公司。

顶级域名（top-level domain，TLD）：域名中的最后一部分，声明了组织的类型或所属国家。

顶级域名	用 途	顶级域名	用 途
.aero	航空业	.jobs	雇佣
.biz	商业	.mil*	美国陆军
.com*	美国商务部	.museum	博物馆
.coop	合作团体	.name	个人和家庭
.edu*	美国教育部	.net*	网络
.gov*	美国政府	.org*	非赢利组织
.info	信息	.pro	专业
.int*	国际组织		

图15-10 一些顶级域名（*代表初始的TLD）

为了缓解域名使用的问题，通过了新的顶级域名集合，它们已经在缓慢推行中。图15-10的右半边展示了新的TLD。这次注册使用新TLD的域名受到了控制，只有具有商标专用权的组织才能申请对应的域名。

域名系统（DNS）主要用于把域名翻译成数字IP地址。在DNS建立前，斯坦福的一个研究小组负责维护一个文件主机表。每建立一个新主机名，斯坦福小组就把它添加到该表（每周两次）。系统管理员会不时读取修改过的主机表，更新他们的**域名服务器**（把主机名翻译（解析）成IP地址的计算机）。

国家代码TLD	国 家
.au	澳大利亚
.br	巴西
.ca	加拿大
.gr	希腊
.in	印度
.ru	俄联邦
.uk	英国

图15-11 基于国家代码的顶级域名

域名系统（domain name system）：管理主机名解析的分布式系统。
域名服务器（domain name server）：把主机名翻译成IP地址的计算机。

随着主机名数量的增长，只用一个表记录主机名已经不可行了，对于更新和分发信息来说，它不是一种实用的方法。1984年，网络工程师设计出了目前使用的复杂域名系统。DNS是一种分布式数据库，没有一个组织要负责更新主机名/IP映射。

当你在浏览器窗口或电子邮件地址中指定了一个主机名时，浏览器或电子邮件软件将给附近的域名服务器发送一个请求。如果这台服务器可以解析主机名，则进行解析，否则这台服务器将把这个请求转发给另一台域名服务器。如果第二台服务器也不能解析它，会继续转发这个请求。最终该请求到达一台能够解析它的服务器，或者因为解析时间太长而过期。

小结

网络是一组连接在一起以共享资源和数据的计算机。网络技术注重的是底层协议和数据传输率。随着我们对网络的依赖性不断增长，出现了客户-服务器模型这种重要的软件技术。

通常根据网络的作用域对它们分类。局域网（LAN）覆盖的是一个小的地理区域以及相对较少的互联设备。广域网（WAN）包括互连网络，把网络连接在一起，覆盖较大的地理区域。城域网（MAN）是专为大型城市设计的。LAN拓扑技术包括环形拓扑、星形拓扑和总线拓扑。以太网已经成了局域网的标准。

开放式系统的基础是一般的网络体系结构模型和协议，具有互通性。OSI参考模型在开放式系统的原则上把网络处理分成了7层。

Internet骨干网是由不同公司提供的一组高速网络。Internet服务提供商（ISP）直接连接到骨干网或连接到其他的ISP，为家用计算机和商业计算机提供网络连接。常用的家庭连接技术包括电话调制解调器、数字用户线路（DSL）和线缆调制解调器。电话调制解调器以音频信号的形式传输数据，因此数据传输速率相当慢。DSL仍然使用电话线，但以数字形式传输数据。线缆调制解调器也是以数字形式传输数据，不过采用的是有线电视的线路。

Internet上传输的消息被分割成了包，每个包将被独立传送到目的地，在此所有包将被重新组合成原始消息。在到达目的地之前，包可能会在网络中进行多次中转。路由器是指导包在网络中的传递路线的网络设备。中继器将在数字信号减弱太多之前强化它们。

网络协议也有分层，这样高层协议将以低层协议为支持。支持Internet通信的关键低层协议是TCP/IP。IP协议和软件负责包的路由。TCP协议和软件负责把消息分割成包以及把包重组为消息，此外还要处理发生的错误。高层协议有SMTP，负责电子邮件通信，FTP负责文件传输，telnet负责远程登录会话，HTTP负责Web通信。一些高层协议具有端口号，用于协助控制和处理网络通信。许多类型的文档和特殊数据格式都有MIME类型。

防火墙可以保护网络免受不正当的访问，给网络施加组织特定的访问控制策略。有些防火墙只会阻止特定端口上的通信，而有些复杂的防火墙则可以分析网络通信的内容。

Internet网络的地址必须精确到一台特定的机器。主机名由易读懂的单词构成，中间由点号分隔。IP地址由四个数字构成，中间由点号分隔，主机名将被翻译成IP地址。IP地址的一部分标识了网络，另一部分标识了该网络中的特定机器。如何划分IP地址，是由该地址引用的网络的类别（A、B或C）决定的。

域名系统（DNS）负责把主机名翻译成IP地址。DNS已经从最初包括所有信息的单个文件发展成了把任务分配给几百万个域名服务器的分布式系统。顶级域名（如.com和.edu）已经变得拥挤不堪了，因此通过了新的顶级域名（如.info和.biz）。

道德问题：无所不在的计算

许多人不知道他们在家或商场享有的隐私权并没有延伸到工作场所。雇员们认为在饮水机旁的谈话或工作时所打的电话都是私密的。通常，他们都错了。虽然他们知道如何保护家里的Internet连接和电话的安全，但是在工作场所却不能给自己提供同样的隐私保护。越来越多的雇主开始采用高科技来监控工作场所。按键程序可以收集并记录一台计算机上的每次按键操作，电话会被监听并录音。有些雇主甚至会安装摄像机和录音装置录下雇员的谈话。有一种软件，如果键盘空置了一段时间，就会启动房间中的视频扫描装置。

美国管理协会每年所做的调查显示，从1997年起，工作场所的侦察措施增加了50%。2003年的调查报告显示，75%的雇主使用了高科技来观察、监视雇员的行动，访问雇员的电子邮件和计算机文件。支持者认为这是个好消息。在求诸法律支持失败后，雇主决定奋起保护自己免受工作场所的不端行为。这是工作隐私权争论的焦点。计算机、电话和物理空间属于雇主，提供给雇员是为了工

作使用。发现员工在网上冲浪，使用电子邮件骚扰他人或与朋友聊天后，雇主发现用同样的技术可以监控雇员是如何使用他们的时间的。雇员Internet监控（EIM）越来越受欢迎。例如，连锁便利店会安装带有录音设备的摄像机，以减少雇员自盗的情况。这种摄像机是由控制中心控制的，所以雇员从不知道自己被监视着。

隐私权支持者则认为这种行为有些过火了。从1997年开始，每年的调查开始后，至少有一个雇主说他的一个雇员滥用了电子邮件。当然，不同的人对滥用电子邮件的定义不同。反对监控技术的人指出，人不是机器。要让老板满意，工作效率更高，他们就需要休息，需要对自己的环境有一定的控制权。如果知道个人电话、走廊中的交谈和电子邮件都被监控了，那么工作场所就会充满怨恨和消极情绪。

管理公平性也不能消除某些人对办公场所监控行为的疑虑。只能访问与业务有关的电子邮件是不能解决问题的，有些人必须阅读其他消息才能进行决策。对于被录下的电话和与业务无关的交谈也有同样的问题。隐私权支持者呼吁制定相关的法规，仅对怀疑的雇员进行监控。直到现在，立法者仍然选择不干涉雇主的监控行为。这些监控措施纯粹出于公司安全的考虑，雇主有权监控工作场所发生的一切。随着技术的不断发展，这些问题也会越来越清楚。

练习

为练习1~6中的定义或空白找出匹配的单词或缩写。

A. LAN B. WAN
C. 网关 D. 总线拓扑
E. 以太网 F. Internet

1. Internet是________。
2. LAN的业界标准。
3. 处理LAN和其他网络之间通信的节点。
4. 连接其他网络的网络。
5. 星形拓扑是一种________配置。
6. 以太网使用的是________。

为练习7~15中的定义或空白找出匹配的单词或缩写。

A. DLS B. TCP/IP
C. UDP D. IP
E. TCP F. 宽带

7. ________和语音通信可以使用同一条电话线。
8. DLS和线缆调制解调器是________连接。
9. 通过常规电话线使用数字信号的Internet连接。
10. 提供的数据传输率一般大于128K bps的网络技术。
11. 把消息分解成包，在目的地再把包组装起来，并且负责处理错误的网络协议。
12. 支持低层网络通信的协议和程序组。
13. TCP的替代者，能够实现较高的传输速率。
14. 处理包路由的软件。
15. ________比UDP可靠。

为练习16～20中的说明或定义找出匹配的协议或标准。

A. SMTP B. FTP
C. Telnet D. HTTP
E. MIME类型

16. 传输电子邮件。
17. 登录远程计算机系统。
18. 把文件传到另一台计算机或从另一台计算机传回。
19. 电子邮件附件的格式。
20. WWW文档的交换格式。

判断练习21～25中的陈述的对错：

A. 对 B. 错

21. 端口是特定高层协议对应的数字标号。
22. 防火墙可以保护局域网不受物理损害。
23. 每个公司都可以建立自己的访问控制策略。
24. 有些顶级域名是注册的组织所属的国家的代码。
25. 两个组织中的计算机不能重名。

练习26～63是问答题或简答题。

26. 什么是计算机网络？
27. 计算机是如何连接在一起的？
28. 节点（主机）指的是什么？
29. 列出并说明与计算机网络相关的两个关键问题。
30. 数据传输率的缩写是什么？

31. 请描述客户/服务器模型，并讨论它如何改变了我们对计算的看法。
32. 局域网到底有多“局部”？
33. 请区分下列LAN拓扑：环形、星形和总线。
34. 拓扑形状如何影响LAN上的信息流。
35. 什么是MAN？MAN和LAN有什么不同？
36. 请区分Internet骨干网和Internet服务提供商(ISP)。
37. 请至少列出两个国家级ISP。
38. 请列出并说明把家用计算机连接到Internet的三种技术。
39. ISP在练习38中的三种技术中扮演什么角色？
40. 练习38中的技术各有哪些优缺点？
41. 电话调制解调器和数字用户线路（DSL）使用同样的电话线来传输数据。为什么DSL比电话调制解调器快很多？
42. 为什么DSL和线缆调制解调器的提供商分配给下载的速度快于分配给上载的速度？
43. Internet上发送的消息将被分割成包。什么是包，为什么要把消息分割成包？
44. 请解释术语包交换。
45. 什么是路由器？
46. 什么是中继器？
47. 包交换会引起哪些问题？
48. 什么是专有系统？它们为什么会引发问题？
49. 我们把多个销售商发售的多平台上的软件和硬件的通信能力叫做什么？
50. 什么是开放式系统？它如何实现互通性？
51. 比较专有系统和开放式系统。
52. 网络交互的7层逻辑分类叫做什么？
53. 什么是协议栈？为什么要把它分层？
54. 什么是防火墙？它能实现什么？是如何实现的？
55. 什么是主机名？它是如何构成的？
56. 什么是IP地址？它是如何构成的？
57. 主机名和IP地址之间是什么关系？
58. IP地址可以分成哪些部分？
59. A类网络、B类网络和C类网络的相对大小是什么？
60. C类网络、B类网络和A类网络中分别可能有多少主机？
61. 什么是域名？
62. 什么是顶级域名？
63. 当前的域名系统如何解析主机名？

思考题

1. 你们学校的计算机系统是什么？所有计算机都连网了吗？网络不止一个吗？宿舍连网了吗？
2. 如果你想注册一个域名，如何申请？.biz、.info、.pro、.museum、.aero和.coop是新的顶级域名。使用这些新的顶级域名有什么限制吗？
3. 你认为Internet这个名字合适吗？Intranet是不是更合适？
4. 利用监控软件跟踪雇员访问Internet和使用电子邮件的雇主是在“侦察”他们的雇员，还是在保护他们重要的商业信息和技术？
5. 是否应该允许雇员出于个人原因在工作时间使用公司的计算机、网络和软件？为什么？

第16章 万 维 网

万维网的发展已经使许多用户开始使用网络通信，如果不是万维网，这些用户恐怕根本不会使用计算机。顾名思义，Web在整个地球上建立了一个像蜘蛛网一样的连接，有了这种基础设施，只要点击一下鼠标按钮，就可以得到想要的信息和资源。是几种不同的基本技术使Web成了今天这种极具价值的工具。这一章将介绍它们中的一部分，建立一个基于Web原则的基础，这是将来所有技术的基础。

目标

学完本章之后，你应该能够：

- 比较Internet和万维网。
- 描述一般的Web处理。
- 编写基本的HTML文档。
- 描述几种HTML标记和它们的用途。
- 描述Java小程序的处理和Java服务器页。
- 比较HTML和XML。
- 定义基本的XML文档和它们对应的DTD。
- 解释如何观看XML文档。

16.1 Web简介

许多人认为Internet和Web这两个词是等价的，事实上，它们有着本质的不同。第15章讨论过计算机网络的一些细节。从20世纪50年代开始，网络就用于连接计算机。虽然Internet已经用于通信多年了，但早期的通信几乎都是采用基于文本的电子邮件和基本的文件交换实现的。

与Internet相比，**万维网**（或简称**Web**）是个相对较新的概念。Web是与使用网络交换信息的软件结合在一起的分布式信息的基础设施。**Web页**是包括或引用各种数据的文档，这些数据包括文本、图像、图形和程序。Web页还包含对其他Web页的**链接**，以便用户能够使用计算机鼠标提供的点击界面随心所欲地到处移动。**Web站点**是一组相关的Web页，这组Web页通常是由同一个人或公司设计和控制的。

> **万维网**（或**Web**）（World Wide Web (or Web)）：信息和用于访问信息的网络软件的基础设施。
>
> **Web页**（Web page）：包含或引用各种类型的数据的文档。
>
> **链接**（link）：两个Web页之间的连接。
>
> **Web站点**（website）：一组相关的网页，通常由同一个人或公司设计和控制。

Internet使通信成为了可能，而Web则使通信变得轻松、更丰富、更有趣。虽然大学和一些高科技公司已经使用了多年Internet，但是直到20世纪90年代中期出现了万维网，Internet才

进入了普通家庭。突然之间，ISP就像雨后春笋一样冒了出来，使人们从家里就能够连接到Internet。Internet成了商业的主要通信工具，很大程度上归功于万维网。电子购物、财务事项往来和小组管理是常见的在线活动。Web已经完全改变了我们的日常生活方式和商业模式。

在使用Web时，我们常常会说“访问”一个Web站点，就像真的到了这个站点一样。事实上，我们只是说明了想要的信息，它们就会呈现在我们面前。访问站点的概念是很容易理解的，因为在进入一个站点之前，我们通常不知道这个站点中有什么。

我们使用**Web浏览器**在Web上通信，如Firefox或Microsoft的Internet Explorer。浏览器是处理Web页的请求并在它到达后将它显示出来的软件工具。图16-1展示了这一过程。

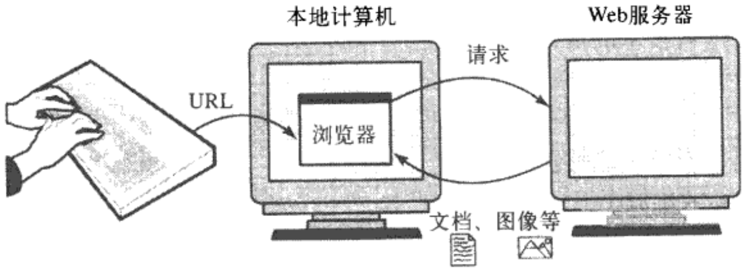


图16-1 浏览器获取一个Web页

被请求的Web页通常存储在另一台计算机上，这台计算机可能就在楼下，也可能在世界的任何角落。用于响应Web请求的计算机叫做**Web服务器**。

在浏览器中，我们用Web地址说明我们想要的Web页，例如：

www.villanova.edu/academics.html

Web地址是**统一资源定位符**（URL）的核心部分，URL唯一标识了存储在世界各处的Web页。注意，URL的一部分是存储信息的计算机的主机名。第15章详细讨论过主机名和网络地址。

Web浏览器（Web browser）：获取并显示Web页的软件工具。

Web服务器（Web server）：用于响应Web页请求的计算机。

统一资源定位符（Uniform Resource Locator, URL）：说明Web地址的标准方式。

除了文本，Web页通常还包括一些独立的元素，如图像。在请求Web页之后，所有与这个页面相关的元素都会被返回。

Web站点的设计和实现技术多种多样。本章的目标是介绍其中几种。在本书的站点上可以找到更多有关这个主题的信息。

Google跳舞的蜘蛛

Google上的高检索量的排名主要集中在在线商务上。Google使用一种蜘蛛工具，每年更新两次或三次排名，这种检索程序会遍历80亿个网页，在在线世界中，被称为Google的舞蹈。专门从事搜索引擎优化的公司（SEO）通过调整Google的蜘蛛工具评估在线商务的站点，来帮助在线商务保持高的搜索排名。SEO公司会简化复杂的网页地址，简化页面上的文本和索引软件要读的看不到的页面描述（叫做元标签）中的关键字。最好的SEO可以极大提高自己客户的检索排名。

16.1.1 搜索引擎

Web*搜索引擎*是帮助你找到其他Web站点的站点。你可能已经多次使用过搜索引擎，如Google或Yahoo。通过输入关键字，说明你想找的信息的类型，搜索引擎就会提供一个有可能满足要求的站点的列表。

搜索引擎是通过检索具有上百万个站点的信息的数据库来生成候选站点列表的。好的搜索引擎会保持自己的数据库是最新的，而且具有匹配关键字和Web页内容的有效技术。

大多数搜索引擎是用用户输入的关键字与作为站点索引的一组关键字进行比较。有些搜索引擎几乎把每个Web页上的每个单词都作为索引存入数据库，只是除去“a”、“an”和“the”这样的常用单词。有些搜索引擎则只用Web页的部分内容作为索引，如文档的标题和副题等。有些索引技术区分大小写，有些则不。

关键字检索非常具有挑战性，因为自然语言（如英语）本身具有二义性（第13章也讨论过这个问题）。例如，术语hard cider（烈性苹果酒）、hard brick（坚硬的砖）、hard exam（很难的测验）和hard drive（磨损的道路）中的hard意思都不同。如果提供了足够的关键字，搜索引擎能够正确地区分匹配站点的优先次序。但在没有上下文的情况下，基本的关键字匹配是很有限的。

有些搜索引擎执行*基于概念的搜索*，即尝试判断所执行的搜索的上下文。如果它们运行得很好，返回的候选页会包含你要检索的主题的相关内容，无论这个页面中的单词是否与查询中的关键字完全匹配。

执行基于概念的搜索的技术有几种。它们通常以复杂的语言理论为基础，这已经超出本书的讨论范围。基本前提是分类，即对比相近的单词。例如，在医学范畴内，心脏这个词可能与动脉、胆固醇和血这些词相近。

基于概念的搜索比关键字检索复杂得多，基于概念的搜索技术很不完善，不过一旦有所改进，这种技术的潜力不可限量。

16.1.2 即时消息

即时消息（instant messaging, IM）应用程序是最受欢迎的Web程序。顾名思义，使用这些程序，你可以实时地给朋友或工作伙伴发送消息。如果发送者和接收者同时运行了即时消息程序，那么消息一到达就会立刻弹出来，这样两个人就能够进行在线交谈。现在领先的IM应用程序是America Online (AOL) Instant Messenger (AIM)。

今天的IM应用程序非常复杂，允许用户定制联系人列表，设置默认的答复，除了文本外，还可以发送标准的图形或订制的图形。IM模式已经成了许多用户的标准通信方法。

大多数IM应用程序采用专有的协议，规定发送消息的格式和结构。AIM的协议也是专有的，但并不限于AOL用户才能使用，这也是AIM如此受欢迎的原因。

即时消息虽然方便，但却不安全。通过各种IM协议发送的消息并没有加密，可能会被网络通信途中的中间点截获。未加密的电子邮件也同样不安全。

16.1.3 博客

Weblog简称为*博客*（blog），是在网站定期发表文章的一种途径。根据发表的作者、主题

和博客的性质，发表的文章可以只是一段，也可以是长篇大论，能够与报纸或杂志上的文章相媲美。

一个网站可以完全被组织成一个博客，也可以把博客作为一个站点的一部分，该站点还可以有其他元素。许多工具和在线服务都使新手能够很容易搭建起自己的博客并发表文章。从20世纪90年代末Weblog出现以来，它已经得到了巨大的改进。虽然我们仍然能看到许多博客发表的是作者自己的无稽之谈和无聊琐事，但还是有许多博客为各种严肃话题提供了出路。有些博客是某种特定问题的重要信息资源，拥有很多追随者。2004年，Merriam-Webster Dictionary宣布博客一词为该年度的重要用词。

有些博主自称公民记者，这就提出了一种新想法，即他们的博客是其他媒体有效且有价值的信息源。一个特定事件标志着这一变化。2004年美国总统大选期间，CBS的Dan Rather报道了一篇文章，文章中的打字错误百出，许多博主都以此质疑这篇文章的真实性。CBS和Rather为文章的真实性据理力争了两个星期，最终还是承认它可能是伪造的。这个事件是网络为普通人提供平台来挑战传统信息的明证。

由于博客是在线发布系统，所以它们对时事的反应比传统的印刷媒体快多了。出于这种原因，许多新闻记者都开辟了自己的博客，以便辅助自己在传统媒体领域的工作。

16.1.4 cookie

cookie是另一种基于Web的技术，增强了Web用户的能力。cookie是Web服务器存储在你的计算机硬盘上的一个小文本文件。站点可能会在用户的机器上存储一个cookie，以捕捉以前这台机器和站点之间发生的交互。

cookie中存储的信息段是名字-值对以及存储信息的站点的名字。例如：

```
UserID   KDFH547FH398DFJ   www.goto.com
```

像这个例子所示的，Web站点可能会为每个访问它的计算机生成一个唯一的ID编号，存储在本地计算机上。更复杂的cookie会存储计时信息，如这台机器访问了站点多久，浏览了哪些内容。

cookie对于Web站点来说用途很多。有些Web站点用cookie来确定有多少不同的访问者。还有些Web站点用cookie存储用户的喜好，以便为用户定制站点的交互。购物车也是用cookie来实现的。

使用cookie的一个问题是人们通常会共用一台计算机来访问Web。由于cookie是基于连接到Web的计算机，而不是基于个人的，所以用cookie个人化站点的访问并不总是行得通。

关于cookie有些常见的误解。cookie不是程序，不会在你的计算机上执行任何操作。它也不能收集有关你或你的计算机的个人信息。这些是常见的误解。不过，由于种种原因，cookie还没有被广泛接受。

16.2 HTML

Web页是用**超文本标记语言**（HTML）创建的。术语*超文本*指的是不像一本书那样线性地组织信息，而是嵌入其他信息的链接，根据需要从一个地方跳转到另一个地方。现在更精确的术语是*超媒体*，因为除了文本之外，我们还要处理很多其他类型的信息，如图像、音频和

视频。

之所以叫做**标记语言**，是因为这种语言的主要元素都是插入文档的**标记**，用于注释存储在该处的信息。在HTML中，这些标记说明了如何显示信息。就像你拿到了一份打印出的文档后，用特殊符号标示一些其他细节一样，如图16-2所示。

> **超文本标记语言**（Hypertext Markup Language, HTML）：用于创建Web页的语言。
> **标记语言**（markup language）：使用标记来注释文档中的信息的语言。
> **标记**（tag）：标记语言中用于说明如何显示信息的语法元素。

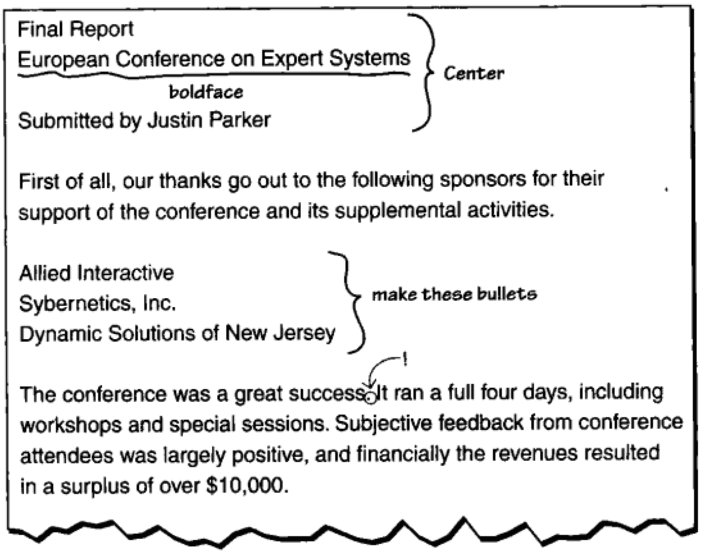


图16-2 一个具有标记的文档

HTML文档是常规的文本文档，用任何一般的编辑器或字处理软件都可以创建它。也有用于协助创建Web页的专用软件，但这些工具最终生成的都是HTML文档。当Web页被请求时，在Web上传输的是这些HTML文档。

HTML的标记说明了信息片段的普通性质（如段落、图像或项目列表）以及如何显示它（如字体、大小和颜色）。可以把标记看作对浏览器的提示。两个不同的浏览器解释同一个标记的方式会稍有不同，因此使用的浏览器不同，看到的Web页也会稍有不同。

让我们看一个浏览器中显示的Web页的例子，然后分析它的HTML文档，看看其中具有的各种标记。图16-3展示了Netscape Navigator中显示的一个Web页。这个页面包含的是一个学生组织Student Dynamics的信息。

这个Web页的顶部有一幅图像，展示了该组织的名字。图像之下是用斜体显示的短语，位于两条水平线之间。短语之下是有关这个组织的信息，包括即将发生的事件的列表和几个短小的段落。最后一个事件结尾处的小图像说明这条信息最近更新过。蓝色的、具有下划线的文本标示链接，用鼠标点击这些链接就可以打开一个新的Web页。注意，有些文本的样式比较特别，如粗体或斜体，有些文本是居中放置的。

图16-4展示了这个Web页的底层HTML文档。它规定了在这个Web中看到的所有格式信息。嵌在主文档内容中的标记用蓝色标识（这里显示不出来）。

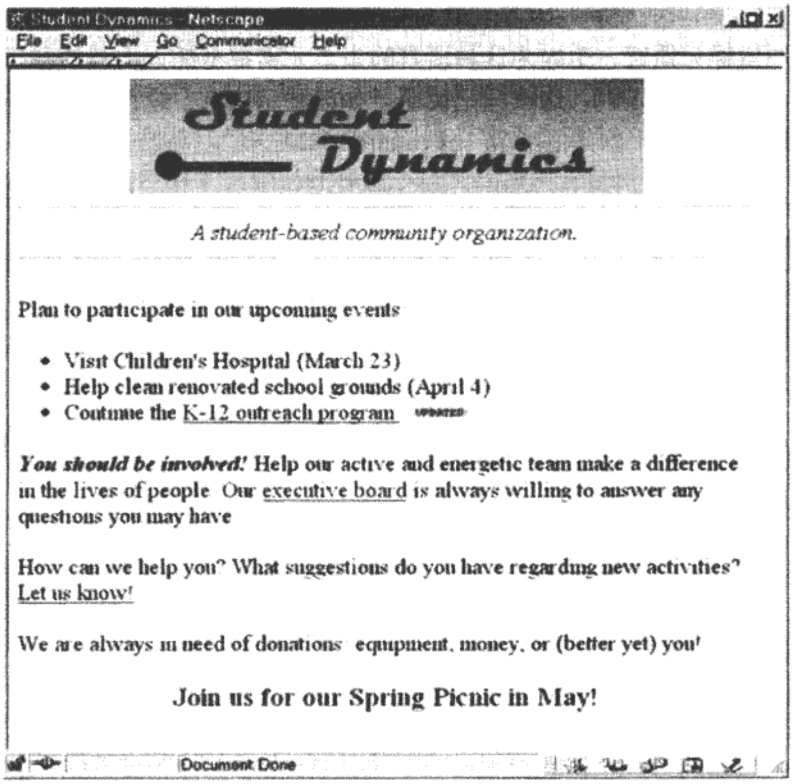


图16-3　Netscape Navigator显示的Student Dynamics组织的Web页

```
<HTML>
  <HEAD>
    <TITLE>Student Dynamics</TITLE>
  </HEAD>
  <BODY>
    <CENTER><IMG SRC="stuDynamics.gif"></CENTER>
    <HR>
    <CENTER><I>A student-based community organization.</I></CENTER>
    <HR>
    <P>Plan to participate in our upcoming events:</P>
    <UL>
      <LI>Visit Children's Hospital (March 23)</LI>
      <LI>Help clean renovated school grounds (April 4)</LI>
      <LI>Continue the <a href="outreach.html">K-12 outreach
          program.</a> <IMG SRC="updated.gif"></LI>
    </UL>
    <P><B><I>You should be involved!</B></I> Help our active and
      energetic team make a difference in the lives of people. Our
      <a href="execBoard.html">executive board</a> is always willing
      to answer any questions you may have.</P>
    <P>How can we help you? What suggestions do you have regarding
      new activities? <a href="suggestions.html">Let us know!</a></P>
    <P>We are always in need of donations: equipment, money, or
      (better yet) you!</P>
    <CENTER><H3>Join us for our Spring Picnic in May!</H3></CENTER>
  </BODY>
</HTML>
```

图16-4　定义Student Dynamics Web页的HTML文档

标记都封装在尖括号中（<...>）。像HEAD、TITLE和BODY这样的单词叫做元素，指定了标记的类型。标记通常是成对出现的，具有一个起始标记（如<BODY>）和对应的结束标记（如</BODY>）。HTML不区分大小写，因此<body>等价于<BODY>。

每个HTML文件都包括两部分，即文档的头和文档主体。文档头包含的是有关文档自身的信息，如文档标题。文档的主体存放的是要显示的信息。

整个HTML文档封装在标记<HTML>和</HTML>中。文档的头和主体是以类似的方式说明的。标记<TITLE>和</TITLE>之间的文本将在页面显示时出现在Web浏览器的标题栏中。

浏览器将根据HTML文档中的标记决定如何显示Web页。它会忽略HTML文档中的格式，如回车符、空格和空行。文档中的缩进只是为了便于人们阅读，与它的最终显示方式无关。浏览器会考虑浏览器窗口的宽度和高度。在调整浏览器窗口的大小后，Web页的内容会被重新格式化，以适应新的窗口大小。

浏览器会尽量搞清楚标记标示文档的方式，从而显示出Web页。如果HTML标记冲突，或者顺序错误，嵌套错误，那么显示的结果会令人吃惊，一点都不美观。

16.2.1 基本的HTML格式

段落标记（<P>...</P>）说明了应该将其中的文本作为单独的段落处理。在大多数浏览器中，结束标记</P>不是必需的，不过为了清楚起见，我们使用它。浏览器通常会用新的一行开始新段落，而且段落前后还有空行，以便与其前后的段落分隔开。

居中标记（<CENTER>...</CENTER>）说明其中的信息应该在浏览器窗口中居中显示。

元素B、I和U分别说明了封装的文本应该用粗体、斜体显示或加下划线。这些元素可以嵌套，从而同时生成多种效果，不过并非所有标记都是如此。也就是说，并非所有元素都能嵌套。

标记<HR>将在页面中插入一条水平线，通常用于把Web页分割成几个部分。

我们通常需要显示项目列表。UL元素标示无序列表，LI元素标示一个列表项。在Student Dynamics这个示例中，标记<UL>...</UL>封装了三个列表项。大多数浏览器都采用项目符号显示无序列表。如果使用有序列表元素（OL），那么列表项将被顺序编号。无序列表和有序列表都可以嵌套，从而创建列表分层。无序嵌套列表的每一层使用的项目符号都不同。有序嵌套列表的每一层都会重新开始编号。

定义文档标题的元素有几种。在HTML中，有6种预定的标题元素，即H1、H2、H3、H4、H5和H6。例如，封装在标记<H3>...</H3>中的文本将被当作3级标题，用比4级标题大、比2级标题小的字号显示。标题标记并非一定要用于设置标题文本，任何想改变字体大小的地方都可以使用它们。

16.2.2 图像和链接

许多标记都具有**属性**，说明了有关信息的额外细节或如何显示封装的信息。

属性的形式如下：

```
属性名=值
```

> **属性**（attribute）：标记中用于提供有关元素的额外信息的部分。

例如，可以用IMG元素把图像嵌入Web页，IMG元素的属性可以标识要显示的图像文件。

属性名是SRC，表示图像的来源。IMG元素没有结束标记。例如：

```
<IMG SRC = "myPicture.gif">
```

这个标记将把图像myPicture.gif插入HTML文档。IMG和SRC之间至少要有一个空格。

在Student Dynamics这个例子中，图像被用作整个页面的标语。在另一个位置，一个小图像被用来说明站点上的这条信息最近更新过。

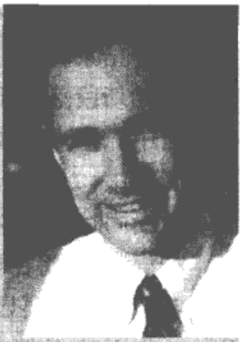
在HTML中，链接是用元素A声明的，A表示锚。该标记的属性HREF指定了目标文档的URL。例如：

```
<A HREF = "http://duke.csc.villanova.edu/docs/">
Documentation Central! </A>
```

这个标记将在屏幕上显示文本“Documentation Central!”，通常这个文本是蓝色的，而且具有下划线。当用户用鼠标点击这个链接时，地址为duke.csc.villanova.edu/docs的Web页将被读取并显示在浏览器中，代替当前的Web页。注意，文件名和URL都封装在引号中。

我们只是简要说明了一下HTML的能力，不过已经介绍过的几个标记足以创建相当丰富有用的Web页了。

Tim Berners-Lee



Tim Berners-Lee是麻省理工学院的计算机科学实验室的第一任3Com（Computer Communication Compatibility，计算机通信兼容性）主席。在麻省理工学院，这是首次由研究人员而不是教员担任主席。与其说Berners-Lee是一位学者，不如说他是一位研究员、传播者和权威人士。他是协调全世界的Web开发的World Wide Web Consortium的指导者。这个联盟与麻省理工学院、法国的INRIA和日本的Keio大学的共同目标是使Web的潜力全部发挥出来，在其高速发展和用途转型的过程中确保它的稳定性。

Tim Berners-Lee是如何得到这个重要职位的呢？当他还是牛津的女王学院的学生时，就创建了自己的第一台计算机。毕业之后，他在Plessey Telecommunications Ltd（英国一家主要的电信设备制造商）工作了两年，然后做了一年半独立顾问，之后在Image Computer Systems Ltd工作了三年。这一时期他做过的项目有实时控制固件、图形和通信软件以及通用宏语言。

1984年，他获得了日内瓦的欧洲核子研究中心（CERN）提供的经费，从事为获取科学数据的异质远程过程调用系统和分布式实时系统以及控制系统的开发。1989年，他提出了一个全球化超文本项目——万维网。人们通过这种技术可以用超文本文档的Web把自己的知识结合起来，从而实现协作。他编写了第一个万维网服务器“httpd”和第一个客户端“World Wide Web”——一个所见即所得的超文本浏览器/编辑器。这项工作开始于1990年10月，同年12月CERN就可以使用程序“World Wide Web”了，1991年夏天，它就开始在Internet上广泛流行起来。

1991年到1993年间，Berners-Lee继续从事Web的设计工作，根据Internet用户反馈的信息对其进行修改。随着Web技术的普及，他最初制定的URL、HTTP和HTML规约都被

细化了。日内瓦的物理实验室显然不适合开发和监管Web。1994年10月，Berners-Lee在麻省理工学院的计算机科学实验室建立World Wide Web Consortium。

在1995年《纽约时报》的一篇访问中，问过Berners-Lee有关统占Web标准以营利的私人公司的问题。他回答道："这种由某个公司统占市场，控制Web标准的危险一直存在。"不过他个人认为不会出现这种情况。"Web的本质是一个全球化的信息源，"他说，"如果把它局限于某个公司，那么它就失去了这种全球性。"

麻省理工学院的计算机科学实验室的主管Michael Dertouzos说过，Berners-Lee先生看来是把自由主义融入了Internet文化。"他承担了保持Web为公众所有的义务，"Dertouzos先生说道，"这是他的使命。"Berners-Lee总结道："合理的竞争会加速创新。公司将会也应该增强它们的浏览器和应用程序的专有性。但是Web导航技术应该是公开的。如果有一天浏览万维网需要6个浏览器，那么万维网就不再是万维网了。"

Berners-Lee曾被美国《时代》杂志评为20世纪100名最重要的人物之一。

16.3 交互式Web页

HTML首次出现时，它那种以有趣的方式格式化基于网络的文本和图像的能力令人震惊。但是，这些信息都是静止的，人们没有办法与Web页中的信息和图片进行交互。

由于用户强烈要求动态的Web，为了满足这些请求，新的技术出现了。这些技术解决问题的方法各不相同。许多新想法都是从新开发的Java程序设计语言衍生出来的，这种语言能够充分利用Web，因为它是独立于平台的。让我们简单地看看这些技术中的两种——Java小程序和Java服务器页。

16.3.1 Java小程序

Java小程序是为嵌入HTML文档而设计的程序，能够通过Web传递给想运行它的用户。Java小程序是在浏览Web页的浏览器中运行的。

> **Java小程序**（Java applet）：为嵌入HTML文档而设计的程序，能够通过Web传输，在浏览器中执行。

Java小程序是用APPLET标记嵌入HTML文档的。例如：

```
<APPLET code = "MyApplet.class" width = 250 height = 160>
</APPLET>
```

当Web用户引用了包含这个标记的页面时，小程序`MyApplet.class`将随其他文本、图像等页面包含的数据被一起发送回来。浏览器知道如何处理每种类型的数据，它将正确地格式化文本，根据需要显示图像。对于小程序，浏览器内置有能够执行小程序的解释器，使得用户能够与之进行交互。Web上有成千上万个Java小程序，大多数浏览器都能够执行它们。

请考虑这种情况内在的困难。在一台计算机上编写的程序将被传递到Web上的另一台计算机上执行。那么如何使在一种类型的计算机上编写的程序在多种类型的计算机上都能够运行呢？关键在于Java程序将被编译成字节码这种程序的低级表示法（如第8章中所提到的），而不是编译成只适用于特定CPU的机器码。任何有效的字节码解释器都能执行字节码，无论

运行字节码的机器是什么类型的。

Java小程序给客户的机器增加了负担。也就是说，Web用户把这些程序带到了自己的机器上，在此执行它们。想起来有些可怕，当你正在网上冲浪的时候，突然某人的程序在你的计算机上运行起来。除非Java小程序只做自己分内的事情，否则这样会带来问题。Java语言具有仔细规划的安全模式。例如，Java小程序不能访问任何本地文件，也不能修改系统设置。

客户的计算机也许能胜任运行小程序的工作，也许不能，这是由小程序的特性决定的。由于这种原因以及小程序是通过网络传输的，所以它们一般都比较小。虽然适用于某些情况，但Java小程序不能完全满足Web用户的交互需求。

16.3.2　Java服务器页

Java服务器页（Java Server Page, JSP）是嵌入了JSP**小脚本**的Web页。所谓小脚本，就是与常规的HTML内容混合在一起的一小段可执行代码。虽然与Java不完全一样，但JSP代码很像一般的Java程序设计语言。

> **JSP小脚本**（JSP scriptlet）：嵌在HTML文档中用于给Web页提供动态内容的代码片段。

JSP小脚本封装在特殊标记<%和%>之间。预定义的特殊对象可以简化某些处理。例如，可以用对象out生成输出，该输出将被融合到Web页中小脚本出现的地方。下面的小脚本将在H3的起始标记和结束标记之间生成短语“hello there”。

```
<H3>
<%
out.println ("hello there");
%>
</H3>
```

这个例子的结果等价于下面的代码：

```
<H3>hello there</H3>
```

不过可以认为JSP小脚本具有完整程序设计语言的强大功能。我们几乎可以利用常规Java程序的各个方面，如变量、条件从句、循环和对象。具备了这种处理能力，JSP页就可以进行重要的决策，生成真正动态的结果。

JSP是在Web页驻留的服务器上运行的。服务器能够在把Web页发送给用户之前决定它的内容。当Web页到达你的计算机时，所有处理都已经完成，生成了（动态创建的）静态的Web页。

JSP尤其适合协调Web页和底层数据库之间的交互。这种类型的处理已经超出了本书的介绍范围，不过在Web上冲浪的时候，你可能会遇到这种处理。电子店铺（主要是为了出售商品而存在的站点）就利用了这种处理方式。有关销售的商品的数据并非存储在静态HTML页中，而是存储在数据库中。当你请求特定商品的信息时，作出响应的可能是一个Java服务器页。这个页面中的小脚本将与数据库进行交互，提取出所需的信息。小脚本和常规的HTML代码将正确地格式化数据，然后把这个页面发送给你浏览。

Wi-Fi标准的重要性

Wi-Fi是便携式计算机现在常用的无线联网技术。为了提高大文件（如电影）的无线传输能力，计算机制造商引入了使用较高Wi-Fi版本802.11n的元件。802.11n把Wi-Fi网络的最高速度从802.11g标准定义的每秒54Mb提高到了270Mb。但是，设计这种元件中的微芯片的技术还没有得到所有设备制造商的一致认可。因此，在高速运行时，这些新设备常常不能与其他设备进行通信。

这种情况是计算技术创新的固有问题的一个实例。公司总是想第一个发布新技术，但它们又想确保跨平台的一致性。标准化至关重要，因为它不仅意味着产品可以一起工作，还意味着随着设备售出的越多，价格会越低，采用这种技术的人也会越多。

16.4 XML

HTML是固定的，也就是说，HTML有预定义的一套标记，每个标记具有自己的语义（含义）。HTML指定了如何格式化Web页中的信息，但是没有说明这些信息表示什么。例如，HTML会说明一条文本的格式是标题，但不会说明这条标题描述的是什么。HTML标记不能描述文档的真正内容。**可扩展标记语言**（XML）允许文档的创建者定义自己的标记集合，从而描述文档的内容。

XML是一种元语言。单词metalanguage（*元语言*）是由单词language（*语言*）加前缀meta构成的，meta的意思是“在……之上的”或“更复杂的”。所谓**元语言**，就是使我们能够精确地运用常规语言的语言，是超出常规语言，用于定义其他语言的语言，就像描述英语规则的语法书。

可扩展标记语言（Extensible Markup Language, XML）：允许用户描述文档内容的语言。
元语言（metalanguage）：用于定义其他语言的语言。

Tim Berners-Lee使用元语言标准通用标记语言（SGML）来定义HTML。XML是SGML的简化版本，用于定义其他标记语言。XML把Web带入了一个新的发展方向。不过XML并没有取代HTML，而是使它更丰富。

与HTML一样，XML文档也是由标记数据构成的。不过在编写XML文档时，不必拘泥于预定义的标记集合，因为根本不存在这样的集合。你可以创建任何描述文档中数据所必需的标记。XML文档的重点不在于如何格式化数据，而在于数据是什么。

例如，图16-5中的XML文档描述了一系列图书。文档中的标记注释了表示每本书的题目、作者、页数、出版商、ISBN和价格的数据。

这个文档的第一行说明了使用的XML的版本。第二行说明了包含该文档的**文档类型定义**（DTD）的文件。DTD是文档结构的规约。该文档剩余的部分是关于两本书的数据。

文档类型定义（Document Type Definition, DTD）：XML文档结构的规约。

特定XML文档的结构是由它对应的DTD文档描述的。DTD文档不只要定义标记，还要说明它们是如何嵌套的。图16-6展示了上例中的XML文档对应的DTD文档。

DTD文档中的`ELEMENT`标记描述了构成XML文档的元素。这个DTD文件的第一行说明books标记由零个或多个book标记构成。在括号中单词book后面的星号（*）表示零个或多

个。接下来的一行说明book标记由其他几个标记按照特定的顺序构成，即title、authors、publisher、pages、isbn和price。下面一行说明authors标记由一个或多个author标记构成。单词author后面的加号（+）表示一个或多个。其他标记被指定为包含PCDATA，即解析过的字符数据（Parsed Character Data），说明这些标记不能再进一步分解为其他标记。

```
<?xml version="1.0" ?>
<!DOCTYPE books SYSTEM "books.dtd">
<books>
<book>
<title>The Hobbit</title>
<authors>
  <author>J.R.R. Tolkien</author>
</authors>
<publisher>Ballantine</publisher>
<pages>287</pages>
<isbn>0-345-27257-9</isbn>
<price currency="USD">7.95</price>
</book>
<book>
<title>A Beginner's Guide to Bass Fishing</title>
<authors>
  <author>J. T. Angler</author>
  <author>Ross G. Clearwater</author>
</authors>
<publisher>Quantas Publishing</publisher>
<pages>750</pages>
<isbn>0-781-40211-7</isbn>
<price currency="USD">24.00</price>
</book>
</books>
```

图16-5　具有关于书籍的数据的XML文档

```
<!ELEMENT books (book*) >
<!ELEMENT book (title, authors, publisher, pages, isbn, price)>
<!ELEMENT authors (author+)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>
<!ELEMENT pages (#PCDATA)>
<!ELEMENT isbn (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ATTLIST price currency CDATA #REQUIRED>
```

图16-6　关于书籍的XML文档对应的DTD文档

这套标记中唯一具有属性的是price标记。DTD文档的最后一行说明了price标记具有一个属性currency，而且是必需的。

XML是组织数据的标准格式，与其他特殊类型的输出无关。一种相关的技术叫做**可扩展样式表语言**（XSL），可以把XML文档转换成特定用户需要的格式。例如，可以定义一个XSL文档，把一个XML文档转换成HTML文档，以便能在Web上看到该文档。还可以定义另一个XSL文档，把同一个XML文档转换成Microsoft Word文档，或转换成适用于PDA（如Palm Pilot）的格式，甚至可以转换成语音合成器使用的格式。图16-7展示了这一过程。本书并不

探讨XSL转换的细节。

> **可扩展样式表语言**（Extensible Stylesheet Language，XSL）：定义XML文档到其他输出格式之间转换的语言。

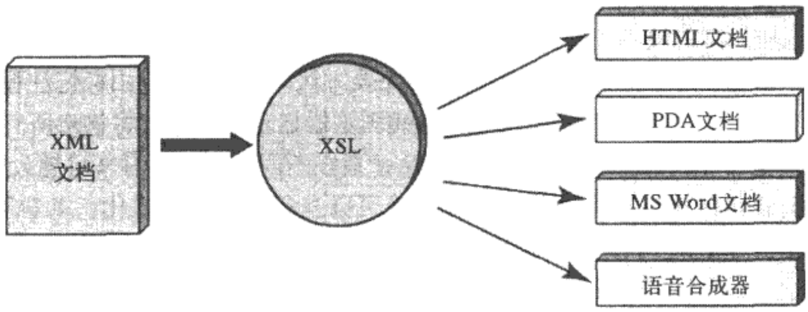


图16-7 一个XML文档可以转换成多种输出格式

用XML规定的语言还有一个方便的特征，即用这种语言编写的文档可以轻松地自动生成。有种软件系统（通常具有底层数据库）可以用来生成大量易于在线传输和分析的数据。一旦生成了，这些数据就能被转换成最适合每个用户浏览的格式。

有些组织为特定的主题开发了专用的XML。例如，化学家和化学工程师定义了化学标记语言（CML）以标准化分子数据的格式。CML包括大量有关化学方面的标记，给化学专业人员提供了共享和分析数据的通用格式。

记住，XML是*标记规约语言*，XML文件则是数据。除非你运行显示XML文件的程序（如浏览器），或者运行用它们进行操作的程序（如把数据转换成另一种格式的转换器或读取数据的数据库），或者运行修改它们的程序（如编辑器），否则什么都不会发生。XML和相关技术为信息管理和以各种方式在Web上有效地进行信息通信提供了强有力的机制。随着这些技术的发展，利用它们的新机会将不断出现。

小结

虽然术语Internet和Web常被混为一谈，但它们并不相同。万维网是分散在世界各处计算机上的信息和访问信息的软件构成的基础设施。Web依靠底层网络（尤其是Internet）在用户之间交换信息。

Web页不仅包含信息，还包含对其他资源（如图像）的引用。由个人或公司管理的一组Web页叫做Web站点。全球各种Web页之间都有链接，这也是万维网这个名字的来源。

所谓访问一个Web站点，其实是请求存储在远程Web服务器上的Web页，把它拿到本地计算机上以便浏览。可以用统一资源定位符（URL）指定我们想浏览的Web文档。

有些Web站点（如google.com）是搜索引擎，用户只要输入单词或关键字，站点就可以根据这些搜索相关信息。搜索引擎会提供一个与用户需求可能匹配的候选站点列表。有些搜索引擎只是以用户输入的关键字为依据，而有些则会尝试解释关键字的内涵。

即时消息（IM）应用程序给了Web另一种交互方式，它允许用户进行在线交谈。IM程序已经发展成了具有图像甚至视频。

Weblog或blog（博客）是定期在网络上发表文章的工具。越来越多的严肃博客成为特定主题的重要信息资源。还有一些博客造就了“公民记者”，他们的工作是对主流媒体的很好补充。

cookie是Web站点存储在你的硬盘上的小文本文件，以便你返回该站点时，该站点能够得到有关你以及你上次访问的信息。它们通常用于跟踪用户的活动，对用户和使用它们的站点都很有帮助。cookie不是程序，因此不能在你的计算机上执行代码。

超文本标示语言（HTML）是定义Web页的主要方法。HTML文档由标记注释的信息构成，标记规定了如何处理和格式化特定的信息。Web浏览器显示HTML时将忽略所有额外的空格、空行和缩进。浏览器完全靠标记指引，同一个Web页在不同浏览器中看来可能会稍有不同。

HTML标记既可以规定整个文档的结构，也可以执行基本的格式化，如标题、段落和居中显示文本等。用标记还可以指定字体样式，如粗体和斜体等。无序列表和有序列表都有自己的标记集合。

有些HTML标记具有属性，声明了额外的信息。例如，图像标记的SRC属性声明了存储图像的文件。锚标记定义了链接，用一个属性声明了目标Web页的位置。

此外，还能够动态地创建Web页。两种支持基于Web的交互的技术是Java小程序和Java服务器页。Java小程序是嵌在HTML页中，由Web浏览器执行的Java程序。它们具有跨平台的特性，因为Java小程序将被编译成Java字节码。

Java服务器页是把小脚本混入HTML代码中，由Web服务器执行，以协助动态地定义Web页的内容。小脚本具有完整语言的强大功能。JSP尤其适用于协调Web页和底层数据库之间的交互。

XML是可扩展标记语言的缩写。XML是一种元语言，即可以用于定义其他语言。HTML标记的重点在于定义显示数据的格式，XML标记则声明了数据的本意。用户不必拘泥于使用特定的标记集合，他们可以定义任何利于描述数据的标记。

XML标记的格式和它们之间的关系定义在文档类型定义（DTD）文档中。XSL（可扩展样式表语言）定义了把XML文档转换成其他用户适用的格式的方法。

道德问题：写博客

像网站一样，博客已经变得无时无刻不在了。博客是一个Weblog，或者说在线杂志。大多数博客具有互动性，读者可以提供反馈意见。虽然大多数博主只是写一些琐事，但还是出现了博客群，成为一种生机勃勃的新闻媒介。博客的影响正在扩大，有时，它的意见成了主流媒体的补充，或者能纠正其中的错误。还有一些博客坚持不懈地对地方和国家新闻提出自己独到的见解。

根据《华尔街杂志》报道，阅读博客的人数正在激增，“阅读博客的美国民众在2004年一跃达到了58%，约3200万人……在2004年的总统大选中，有110万人阅读谈论政治的博客中的新闻”。但是博客不仅仅适用于在线记者或政治评论员。它的应用还扩展到了医生、律师和教师。甚至在教室中，博客也开始日益盛行。许多学生都有自己的博客，他们在其中用日记的形式记录自己对老师的印象或者与其他学校相关的信息。学生对博客的使用引发了新的争论，即教育者对学生在在线教室中的活动有多大的控制权呢？[1]

当然，博客群自身也不乏争议。2005年，一些博主贴出了关于Apple公司未发布的Apple计算机的一些机密文档，从而引爆了一场争论。Apple公司坚决要求知道这些信息的来源，但是博主则坚称他们是记者，应该受保护，不必揭示信息源。但是加州的一位法官并不同意这些辩解，他裁决博主必须公布信息源。不幸的是，这位法官并未解决核心问题，即博主是否享有与记者一样的权益

来保护自己的信息源。一方面，这些博主确实像记者一样报道新闻，那么他们为什么不能享有和记者一样的权益呢？另一方面，未来可能有1000万、2000万或者5000万博主，法官和律师都担心他们全部申请记者权益，他们害怕拥有网站的人要求陪审团作证，拒绝合作。

由于写博客是一个新现象，所以关于“写博客的道德”还没有太多的争论。但这样的争论是必需的。博主的责任是什么？尤其是那些对新闻站点有影响的博客。他们有与传统媒体一样的义务吗？他们是否要遵守同样的客观标准？[2]

虽然对博主进行太多限制不是个好办法，但他们的确应该像其他传递信息的人一样尽道德义务。首先，博主不能撒谎。St. Thomas Aquinas把谎言定义为故意说错的话。[3]以Aquinas的观点，谎言是可憎的，因为它冒犯了真理，破坏了日常生活必需的和谐。从法律角度来看，谎言和欺骗是错误的，因为它们掩盖了知识本来的价值。因此，像其他人一样，博主必须一直保持信息的真实性。此外，他们还有义务不时地检查信息源，鉴别信息源，确保读者能完全得到信息。在在线环境中，提供其他网站的链接就可以做到这一点。博主还要避免发布不正当的指责，尽可能快地取消错误信息。最后博主应当公布那些有可能影响客观性的利益冲突。有时，博主有必要公布谁付给他工资，或者谁为他的网站运作提供资金。“读者在访问你的站点时应该可以假设你没有接受贿赂”。[4]如果博主可以遵守这些简单的规则，那么读者就会对他们产生信任，Weblog就会有一个光明的未来。

此案例中引用的资料摘自R. Spinello, Cyberethics: *Morality and Law in Cyberspace, 3rd edition* (Sudbury, MA: Jones and Bartlett, 2006)。

练习

判断练习1～12中的陈述的对错：

A. 对　　B. 错

1. Internet和Web本质上是同一个事物的两个名字。
2. 响应Web请求的计算机是Web浏览器。
3. 访问Web站点实际上是把站点拿到我们的计算机上。
4. 大多数搜索引擎使用基于上下文的方法查找候选页。
5. Weblog就是博客。
6. 公民记者可以用Weblog在线发布文章。
7. cookie是在你的计算机上执行的程序。
8. 在请求一个Web页后，所有与它相关的元素都将被带到你的计算机上。
9. 从20世纪50年代起就开始使用网络连接计算机了。
10. 直到Web出现才有了网络通信。
11. Web是在20世纪90年代中期出现的。
12. 要访问Web必须有Web浏览器。

为练习13～22中的定义或空格找到匹配的单词或缩写。

A. JSP小脚本　　B. URL

C. HTML　　D. 标记

E. Java小程序　　F. XML

13. 设计用于嵌入HTML文档的程序。
14. 每个Web页的唯一标识。
15. ______是在Web服务器上运行的。
16. ______是在Web浏览器上运行的。
17. ______的标记是固定的。
18. ______的标记不是预定义的。
19. ______是一种元语言。
20. ______文档的结构是由对应的DTD描述的。
21. 标记语言中的语法元素，说明了如何显示信息。
22. ______的一部分是存储信息的计算机的主机名。

练习23～70是问答题或简答题。

23. 什么是Internet？
24. 什么是Web？
25. 什么是Web页？
26. 什么是Web站点？
27. 什么是链接？
28. 为什么把万维网比喻成蜘蛛网？

29. Web页和Web站点之间是什么关系？
30. Internet和Web之间有什么区别？
31. 请描述Web用户如何获取并浏览一个Web页。
32. 什么是统一资源定位符（URL）？
33. 什么是标记语言？这个名字的来源是什么？
34. 请对比超文本和超媒体。
35. 请说明HTML的语法。
36. 什么是水平线？它有什么作用？
37. 请列举5种用HTML标记建立的格式规约。
38. 什么是标记的属性？请举例。
39. 请编写一个HTML语句，把图像mine.gif嵌入Web页。
40. 请编写一个HTML语句，建立链接http://www.cs.utexas.edu/users/ndale/，并在屏幕上显示文本“Dale Home Page”。
41. 如果用户点击了练习40中建立的链接“Dale Home Page”，会出现什么情况？
42. 为你学校的某个组织设计并创建一个HTML文档。
43. 创建一个或多个HTML文档，说明你的个人喜好。
44. 什么是Java小程序？
45. 如何把Java小程序嵌入HTML文档？
46. Java小程序是在哪里执行的？
47. 对Java小程序有哪些限制？为什么？
48. 什么是Java服务器页？
49. 什么是小脚本？
50. 如何把小脚本嵌入HTML文档？
51. JSP处理与小程序处理有哪些不同？
52. 什么是元语言？
53. 什么是XML？
54. HTML和XML有哪些相同点和不同点？
55. XML文档与文档类型定义之间有什么关系？
56. a) 在DTD中，如何说明一个元素要重复出现零次或多次？
    b) 在DTD中，如何说明一个元素要重复出现一次或多次？
    c) 在DTD中，如何说明一个元素不能再分解成其他标记？
57. 什么是XSL？
58. XML和XSL之间有什么关系？
59. 如何浏览XML文档？
60. 为你的课程定义XML语言（DTD），然后生成一个示例XML文档。
61. 为政府机关定义XML语言（DTD），然后生成一个示例XML文档。
62. 为动物园的动物定义XML语言（DTD），然后生成一个示例XML文档。
63. 本章有很多缩写。请定义下列缩写。
    a) HTML　b) XML
    c) DTD　d) XSL
    e) SGML　f) URL
    g) ISP
64. 为具有下列特性之一的Web页创建HTML文档。
    a) 居中的标题　b) 无序列表
    c) 有序列表　d) 链接到另一个Web页
    e) 图片
65. 请区分HTML标记和属性。
66. 为什么同一个Web页在不同的浏览器中看来有所不同？
67. 每个HTML文档都具有哪两个部分？
68. HTML文档的两部分的内容是什么？
69. 在声明Web页的URL的标记中，A表示什么？
70. 为具有下列特性之一的Web页创建HTML文档。
    a) 用大字体靠右对齐显示的标题
    b) 名为Exercise.class的小程序类
    c) 两个不同的链接
    d) 两张不同的图片

思考题

1. Web对你个人有什么影响？
2. 在上这一课之前，你有自己的Web站点吗？它有多复杂？你使用的是HTML还是其他Web设计语言？如果你使用的是其他语言，请查看你的Web页的源代码，看看真正格式化Web站点的HTML标记。其中有什么是本章没有介绍的吗？如果有，请查找它们的含义。（在哪里查找？当然是在Web上。）

3. 你曾经上过采用Web教学的课程吗？你喜欢这种方式吗？你认为这样学到的东西比常规课程学到的多还是少？
4. 请想象一下Web的未来。
5. 以你的观点，对于博主应该施加多大的制约？应该与记者的标准一样吗？
6. 博客是与普通大众交流的有效工具吗？还是说博客表达的是个人观点，由于没有编辑，所以是不可信的信息源？
7. 博客只对新闻报道有用吗？还是有其他的价值？

第八部分 总 结

第17章 计算的限制

前16章介绍了什么是计算机，它们能够做什么，以及如何用它们解决问题。这一章将介绍计算机不能做什么。也就是说，我们将分析硬件、软件和问题自身强加于计算机的限制。字典对“限制”这个词有很多解释，其中有“界限”和“令人恼怒的或无法忍受的事物”这两种意思。这一章所指的“限制”包括这两重意思。

就像路障会阻断交通一样，这些硬件、软件和问题带来的限制也阻止了某些类型的处理。

目标

学完本章之后，你应该能够：

- 说明硬件给计算问题的解决方案强加的限制。
- 讨论计算机的有限性对数字问题的解决方案造成了哪些影响。
- 讨论一定能探测出数据传输中的错误的方法。
- 说明软件给计算问题的解决方案强加的限制。
- 讨论构建更好的软件的方法。
- 说明计算问题自身固有的限制。
- 从P类问题到不能解决的问题，讨论问题复杂度的连续性。

17.1 硬件

硬件带给计算的限制来自于几个因素。其一，数字是无限的，而计算机的数字表示却是有限的。其二，硬件就是硬件，也就是说，它是由易坏的机械部件和电子部件构成的。其三，在把数据从一个内部设备传递给另一个内部设备，或者从一台计算机传递到另一台计算机时会发生问题。让我们来看看每种问题和最小化它们的影响的一些策略。

17.1.1 算术运算的限制

第2章和第3章讨论过数字和它们的计算机表示法。计算机的硬件对整数和实数的表示法都有限制。

整数

在第7章讨论过的Pep/7中，进行算术运算的寄存器是16位的。我们说过，如果只表示正数，它能存储的最大值是65 535，如果既要表示正数，又要表示负数，它能存储的最大值是32 767。Pep/7是一台虚拟机，那么真正的计算机会怎样呢？如果计算机的字长是32位，那么它能表示的整数范围是－2 147 483 648到2 147 483 647。有些硬件系统支持长字算术，范围是－9 223 372 036 854 775 808到9 223 372 036 854 775 807，这样的长度足够进行任何运算吗？

Henry Walker在他的著作《The limits of Computing》中讲了这样一个故事。[1] 一位国王请一个年轻聪明的姑娘为他办一件事，她说如果有丰厚的报酬，就帮国王做这件事。她给了国王两个选择，一个是把今后5年这个王国生产的粮食的五分之一付给她，一个是使用棋盘给她报酬，规则如下：

- 棋盘的第一格放1粒稻谷。
- 棋盘的第二格放2粒稻谷。
- 棋盘的第三格放4粒稻谷。
- 棋盘的第四格放8粒稻谷。
- 每个后继方格中的稻谷数量是前一格的两倍，直到64个棋盘格放完为止。

经过一阵思考，国王选择了第二种支付方案。(你会选哪个呢？)

当要把报酬给这个姑娘时，国王开始在棋盘格中摆放稻谷。第一行有255（1+2+4+8+16+32+64+128）粒稻谷，他想，“还好”。第二行有65 280粒稻谷，还不太糟糕。但是，第三行有963 040粒稻谷，使国王感到了不安。在计算第四行的稻谷数时，国王先计算了棋盘最后一格的数量，现在他明白这个模式了。只是第64个格子就有2^{63}粒稻谷，约为8×10^{18}粒，相当于110 000亿蒲式耳。国王欠了这么大笔债，只好退位，精通数学的姑娘就成了女王。

这个故事告诉了我们，整数可以增长得非常快，增长到非常大。如果计算机字长是64位，只表示正数，那么最多只能表示第64个棋盘格中的稻谷数。如果想把64个棋盘格中的稻谷数加起来，我们就做不到了。这样将会发生溢出。

计算机的硬件决定了它能表示的数字（整数和实数）的限制。不过用软件方法可以克服这种限制。例如，可以用一系列较小的数表示很大的数。图17-1展示了如何通过在每个字中放一位数字表示整数。

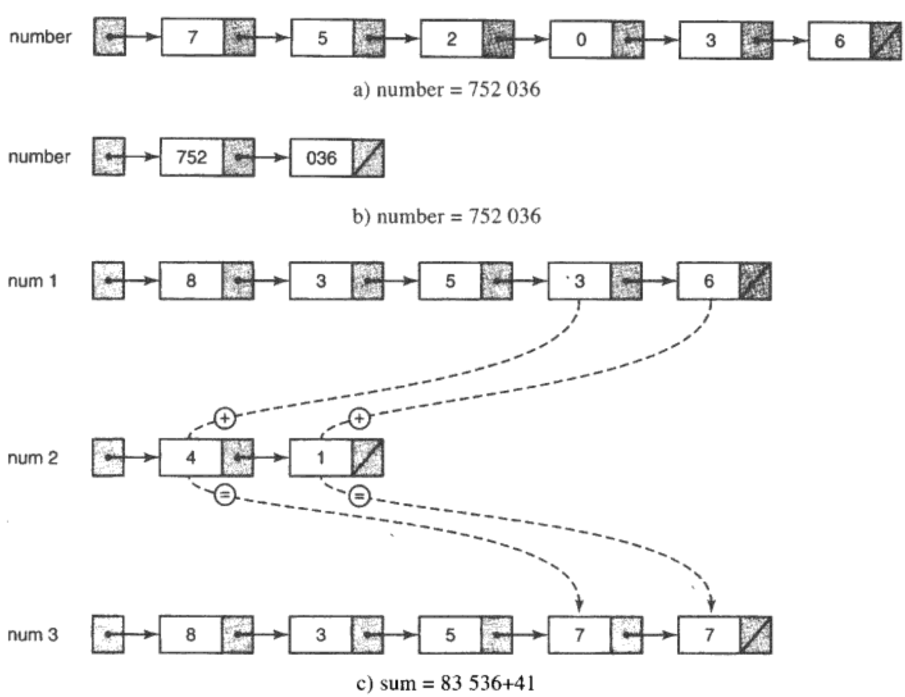


图17-1 表示非常大的数

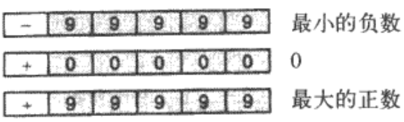
操作这种形式的整数的程序必须从最右边开始把每个数对相加，并且把进位加到左边一位的加法中。

实数

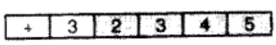
第3章介绍过，实数被存储为整数加说明小数点位置的信息。为了更好地理解为什么实数会带来问题，让我们看一个表示数字和小数点信息的编码模式。

为了便于讨论，我们假设计算机的内存单元大小相同，每个内存单元由一个符号和5个数字位组成。每当定义了一个变量或常量，赋予它的内存单元都由5个数字和一个符号构成。如果定义的是整数变量或常量，那么这个数会被直接存储起来。如果声明的是一个实数变量或实数常量，那么这个数将被存储为整数部分和小数部分，要表示这两部分，必须对该数编码。

让我们来看看编码后的数是什么样的以及这些编码如何表示程序中的算术值。我们从整数开始。用5位数字能够表示的整数范围是－99 999到＋99 999：



精度（最多可以表示的位数）是5个数位。这个范围内的每个数都能被精确表示出来。如果用其中一个数位（如最左边的一位）表示指数会出现什么情况呢？例如：



表示数字$+2345\times10^3$。现在，我们能表示的数的范围大得多了：

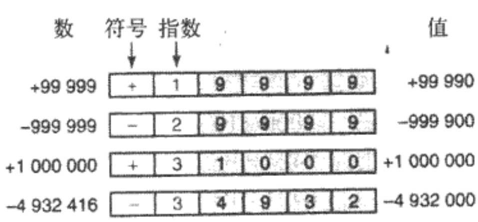
-9999×10^9到$+9999\times10^9$

或

－9 999 000 000 000到＋9 999 000 000 000

> **精度**（precision）：最多可以表示的有效位数。

现在精度只有4位数字。也就是说，我们只能表示每个数中的4位**有效位**（非零数字或纯粹的零）。这意味着这个系统只能精确地表示4位数。对于更大的数会出现什么情况呢？最左边的4位数字是正确的，其余的数字都被假设为0。右边的数位或者说最低有效数位将丢失。下面的例子说明了这种情况：



> **有效位**（significant digits）：从左边的第一个非零数位开始，到右边的最后一个非零数位（或纯粹的零）结束的数字。

注意，我们只能精确地表示1 000 000，但不能精确地表示－4 932 416。我们的编码模式

仅限于4位有效位，不能表示的数字被假设为0。

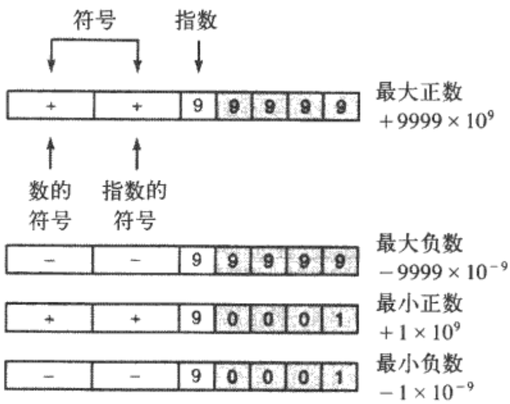
要扩展这种编码模式来表示实数，还要能够表示负指数。例如，

$$4394 \times 10^{-2} = 43.94$$

或

$$22 \times 10^{-4} = 0.0022$$

由于在我们的模式中，指数没有符号，所以必须对它稍加修改，把已经有的符号作为指数的符号，再在这个符号的左边加一个符号，作为数本身的符号。



现在我们可以表示-9999×10^{-9}到9999×10^{9}之间的所有数（精确到四位）了，包括所有的小数值。

假设我们想用这种编码模式求三个数x、y与z的和。可以先求x与y的和，再把z加到之前求得的结果上。也可以先求y与z的和，再把x加到之前求得的结果上。算术运算中的结合律可以证明这两种方法得到的答案一样，但结果真是这样吗？

计算机限制了实数的精度（有效位的位数）。让我们用4位有效位加一位指数的编码模式求下列三个值的和：

$$x = -1324 \times 10^{3} \quad y = 1325 \times 10^{3} \quad z = 5424 \times 10^{0}$$

把z加到x与y的和上的结果如下：

$$\begin{array}{ll} (x) & -1324 \times 10^{3} \\ (y) & \underline{\ \ 1325 \times 10^{3}} \\ & \quad\ 1 \times 10^{3} = 1000 \times 10^{0} \end{array}$$

$$\begin{array}{ll} (x+y) & 1000 \times 10^{0} \\ (z) & \underline{5424 \times 10^{0}} \\ & 6424 \times 10^{0} = (x+y)+z \end{array}$$

把x加到y与z的和上的结果如下：

$$\begin{array}{ll} (y) & 1\,325\,000 \times 10^{0} \\ (z) & \underline{\quad\quad 5424 \times 10^{0}} \\ & 1\,330\,424 \times 10^{0} = 1330 \times 10^{3}\text{（截取4位数字）} \end{array}$$

$$\begin{array}{ll} (y+z) & 1330\times10^{3} \\ (x) & \underline{-1324\times10^{3}} \\ & 6\times10^{3}=6000\times10^{0}=x+(y+z) \end{array}$$

这两个答案的千位上的结果相同，但百位、十位和个位上的结果却不同。这叫做**表示误差或舍入误差**。y与z的和是精度为7位的数，但是只有4位被保存了下来。

除了表示误差，浮点算术还有两个要注意的问题——下溢和溢出。当计算出的绝对值太小以至于计算机不能表示时，将发生**下溢**。采用十进制数表示法，让我们看一个涉及非常小的数的运算：

$$\begin{array}{r} 4210\times10^{-8} \\ \underline{\times2000\times10^{-8}} \\ 8\ 420\ 000\times10^{-16}=8420\times10^{-13} \end{array}$$

用我们的编码模式不能表示这个数，因为指数－13太小了。我们的最小指数是－9。因此，这个运算的结果将被设为0。所有因为太小而不能表示的数都将被设为0。在这种情况下，这样做是合理的。

当计算出的绝对值太大以至于计算机不能表示时，将发生**溢出**。溢出是更加严重的问题，因为一旦发生溢出，没有合理的解决方法。例如，下列计算的结果

$$\begin{array}{r} 9999\times10^{9} \\ \underline{\times1000\times10^{9}} \\ 9\,999\,000\times10^{18}=9999\times10^{21} \end{array}$$

不能存储。我们应该怎么处理呢？要与下溢的处理保持一致，可以把结果设置为9999×10^{9}，即模式中的最大实数值。但凭直觉就能看出这是不对的。另一种方法是停止运算报错。

> **表示（舍入）误差**（representational (round-off) error）：由于算术运算结果的精度大于机器的精度造成的算术误差。
>
> **下溢**（underflow）：当计算的结果太小以至于给定的计算机不能表示时发生的情况。
>
> **溢出**（overflow）：当计算的结果太大以至于给定的计算机不能表示时发生的情况。

浮点数可能发生的另一种错误叫做**化零误差**。当相加或相减的两个数的量级相差太大时会出现这种误差。下面是一个例子：

$$(1+0.000\ 012\ 34-1)=0.000\ 012\ 34$$

算术运算的法则可以证明这个等式是正确的。但如果用计算机来执行这个运算会出现什么情况呢？

$$\begin{array}{r} 100\,000\,000\times10^{-8} \\ \underline{+1234\times10^{-8}} \\ 100\,001\,234\times10^{-8} \end{array}$$

因为只有4位精度，所以结果将变为1000×10^{-3}。计算机再减去1：

$$\begin{array}{r} 1000\times10^{-3} \\ \underline{-1000\times10^{-3}} \\ 0 \end{array}$$

结果是0，而不是0.000 012 34。

> **化零误差**（cancellation error）：由于精度限制，当相加或相减的两个数的量级相差太大时发生的精确度损失。

我们已经讨论过实数的问题，整数（无论负数还是正数）也会发生溢出。这一节讨论的主旨有两点。第一，实数运算的结果通常与你预期的不同。第二，如果处理的数非常大或非常小，要注意执行运算的顺序。

17.1.2 部件的限制

"我的硬盘坏了。""文件服务器崩溃了。""我的电子邮件昨晚被破坏了。"任何计算老师都听到过几百次这样的抱怨，这是学生为迟交作业所做的解释（或借口）。当然，如果提前交了作业，这些问题就都能克服了。但是，硬件故障的问题确实存在，硬盘确实会坏，文件服务器确实会崩溃，网络也确实会断。J.A.N.Lee杜撰了Titanic效应这个词，用来形容系统崩溃的严重程度超出了设计者的想象力。[2] 硬件故障确实会发生，最后的解决方法是进行防御性维护。在计算领域，这意味着定期检测硬件的问题，替换损坏的零件。

防御性维护还要保证放置计算机的物理环境合适。大型计算机常常需要具有空调和无尘的房间。PC不能放在防漏水管下。唉，并非所有的情况都能预计到。在集成电路出现之前就发生过这种情况。一台运行正常的计算机却开始生成奇怪的结果。最后才发现是只蛀虫进入机箱造成的。从此以后，术语"bug"就表示计算机错误。最近的一次事故是有关DSL线的，它会间歇性地自我中断。最后才发现，问题出在电话线上，松鼠用它来磨牙了。

当然，关于部件限制的所有讨论都有一个前提，即计算机硬件在设计和制造阶段都经过了全面的测试。1994年有一条关于Intel的Pentium处理器的电路缺陷的丑闻。IBM、Compaq、Dell、Gateway 2000等公司生产的几百万台计算机都使用了Pentium芯片。这个电路缺陷是浮点部件的一个设计错误，会使5位有效位的某些除法运算生成错误的答案。

这个错误会多久影响一次运算呢？IBM预测，电子制表软件的用户将每隔24天遭遇一次这样的错误。Intel则声称每隔27 000年才会发生一次错误。PC Week的测试组得出的结论是发生错误的频率为2个月到10年一次。[3] 虽然这种芯片被修正了，但Intel公司并没有招回有缺陷的芯片。对Intel公司来说，这是公共关系的灾难，不过直到今天为止，Intel仍是领先的芯片制造商之一。

17.1.3 通信的限制

计算机之内和计算机之间的数据流是计算的生命血液。因此，一定要保证数据不被破坏。实现这一点的策略叫做*检错码*和*误差校正码*。检错码可以判断出在数据传输过程中是否发生了错误，并警告系统。误差校正码不仅能检测出发生的错误，还能判断出正确的值是什么。

校验位

校验位用于检测存储和读取或发送和接收一个字节的过程中发生的错误。校验位是在使用这种模式的硬件中的每个字节上附加了一个位。这个位用于确保9位数值（一个字节加一个校验位）中的1的个数是奇数（或偶数）。

*奇数奇偶校验*要求一个字节加一个校验位中有奇数个1。例如，如果一个字节中的值是

11001100，那么校验位是1，这样才能得到奇数个1。如果字节中的值是11110001，那么校验位是0。当从内存中读取或接收了一个字节时，将计算其中1的个数（包括校验位）。如果1的个数是偶数，说明发生了错误。如果硬件采用这种模式，每个字节将多一个附加位，只有硬件才能访问这个位，用于检测错误。偶数奇偶校验的模式与奇数奇偶校验的相同，只是其中必须有偶数个1。

校验数位

上述模式的一种软件变体是求一个字节中的每个数位的和，然后把和的个位与字节中的数存储在一起。例如，对于数字34376，每个数位的和是23，因此存储的数就是34376－3。如果这个数中的4变成了3，就可以检测到错误。但是，如果7变成了6，而6变成了7，那么数位的和仍然是正确的，但是数却是错的。

这种模式可以扩展为多一个附加位，可以是奇数位的和的个位数。例如，34376可以存为34376－23，3是所有数位和的个位数，2是第1位、第3位和第5位的和的个位数。这种方法能捕捉到相邻数位之间的传输错，但却会漏掉其他的传输错。当然，还可以存储偶数位的和的个位数。也就是说，要检测的错误越重要，检测算法就越复杂。

误差校正码

如果对于一个字节或一个数保存了足够的信息，那么可以推导出错误的数位应该是什么。极端的冗余是对每个存储的值都保留两个独立的副本。如果发现奇偶校验或校验数位有错，那么可以查阅另一个副本以得到正确的值。当然，两个副本可能都有错。

误差校正码主要用于硬盘驱动器或CD，CD表面的不完整性会破坏数据。

航空软件的问题

老式飞机使用的软件还不到100万行代码，而商用飞机的最新软件已经有500多万行代码了。因此，即使商用飞机使用的软件经过的准备和检查再严格，与日常使用的小程序相比，在投入飞行使用前，要发现其中的问题也变得越来越困难了。航空软件中的一个bug造成的后果，比计算机崩溃或者数据丢失这样的问题严重多了。近来的一起案例是：从澳大利亚的Perth飞往马来西亚的Kuala Lumpur的一架飞机，突然在3000英尺的高空陡直上升。机长断开了自动驾驶系统，使机头向下才避免了飞机失速，但是这样导致了陡降。他立即关闭了两个发动机，试图降低飞机的速度。然而，飞机又开始爬升。最后，整个机组又得到了控制权，通过手动操作，安全地把乘客送回了澳大利亚。之后，调查人员发现一个有缺陷的程序向飞行计算机提供了错误的飞行速度和高度的数据，而且计算机没能对飞行员的终止指令立即作出响应。

最初开发自动驾驶系统是为了接管常规任务，以便飞行员能集中注意力，使飞行更顺利。自动驾驶系统的优点使空中旅行变得更加安全。在过去20年中，商用飞机的事故率大大降低了。在美国，20世纪80年代末，每100万架次中有1.3次坠毁事故。到2005年为止，这个平均数字为每100万架次中有0.4次坠毁事故，这意味着200万架次的飞行中，坠毁的飞机数小于1。

17.2 软件

我们都读到过有关具有错误的软件的可怕故事，这些故事听来很有趣。那么正在运行的程序中的软件错误真的经常发生吗？难道没有办法使软件错误更少一些吗？为了回答第一个问题，我们进行了Web搜索，关键字是“software bugs”，得到的相关条目有261 000 000条之

多。软件开发者们正在致力于解答第二个问题。在下面的几节中，我们将分析为什么开发没有错误的软件很困难，也将分析当前软件质量的方法，最后还将给出一组有趣的bug。

17.2.1 软件的复杂度

如果接受“商业软件具有错误”的前提，那么逻辑问题就是“为什么？”难道软件开发者不测试他们的产品吗？这种问题并非是由懒惰引起的，而是由软件的复杂度引起的。随着机器的功能变得越来越强大，计算机能够解决的问题也变得越来越复杂。以前一个问题由一个程序员解决，现在成了一个问题由一组程序员解决，最后一个问题会由一组程序员组解决。

软件测试能够证明存在bug，但是不能证明不存在bug。我们可以测试软件，发现问题，修正问题，然后再测试软件。随着我们不断发现问题，解决问题，对软件的信心也会逐渐增强。但我们永远不能确保已经除去了所有的bug。软件中潜伏着其他的bug，我们还没有发现，这种可能性将一直存在。

由于我们永远不知道是否已经发现了所有问题，那么何时才能停止测试呢？这成了一个风险问题。如果你的软件中还有bug，那么你愿意承担多大的风险？如果你在编写游戏，那么面对的风险是别人捷足先登了，如果你编写的是飞机控制软件，那么要承担的风险就是整机乘客的性命。

Nancy Leveson在《Communications of the ACM》中指出过，20世纪60年代出现的计算分支*软件工程*的目标就是把工程原则引入软件开发。[4] 在过去的半个世纪中，这方面的研究已经向目标跨进了一大步，包括对抽象的角色更深理解、模块性的引入以及软件生命周期的概念。后面将详细介绍它们。

虽然大多数概念来自工程学，但它们必须适合处理更抽象的数据时会发生的特殊问题。硬件设计受实现设计所用的材料的指导和限制。软件则主要受人类能力的限制，而不是物理限制。Leveson博士还指出过，“因此，前50年的特征是学习这个领域的限制，这与人类能够处理的复杂度的限制息息相关。”

构建软件的重点已经变了。以前是构建新软件，而今天，现有软件的维护和升级的问题越来越多，逐渐占据了中央舞台。随着系统变得越来越大而且需要整组的设计员，我们必须开始分析人类协作的方式，以便设计出能辅助人们有效协作的方法。

17.2.2 当前提高软件质量的方法

虽然不可能使大型软件系统完全没有错误，但是并不意味着我们应该放弃。我们可以采用某些策略来提高软件的质量。

软件工程

第6章列出了计算机问题求解的三个阶段，即开发算法、实现算法和维护程序。如果从定义明确的小任务转移到大型的软件项目，那么还需要增加两个阶段，即*制定软件需求和规约*。**软件需求**是用概括而精确的语句列出软件产品提供的功能。**软件规约**则详细说明了软件产品的功能、输入、处理、输出和特性。软件规约说明了程序能够做什么，而不是怎么做。

软件需求（software requirement）：说明计算机系统或软件产品提供的功能的语句。

软件规约（software specification）：软件产品的功能、输入、处理、输出和特性的详细说明。它提供了设计和实现软件所必需的信息。

Leveson博士把软件生命周期看作软件工程要规划的一部分。所谓软件生命周期指的不仅是编码，而是软件的开发和升级。因此，生命周期包括下列阶段：

- 需求分析
- 制定规约
- 设计（高层和低层）
- 实现
- 维护

所有阶段都要执行验证操作。需求是否精确反映了需要的功能？规约是否精确反映了满足需求所需的功能？高层设计是否精确反映了规约中的功能？设计中的每个后继层是否精确实现了上一层的功能？代码实现是否与设计相符？维护阶段实现的改变是否精确反映了想要的改变？这些改变的实现是否正确？

第6章到第8章讨论了一些小问题的设计和代码的测试。显然，随着问题的增大，验证操作也会越来越重要，越来越复杂。虽然设计和代码的测试是整个过程很重要的一部分，但也只是一小部分。在一个典型的项目中，有一半错误是在设计阶段发生的，而实现阶段发生的只是一半错误而已。这个数据会引起一些误解。如果以修正错误的代价为衡量标准，那么在设计过程中越早发现错误，修正错误的花费越小。[5]

大型软件产品是由程序员组制作的。程序设计小组使用的两种有效验证方法是*走查*和*审查*。（虽然第6章已经介绍过这两种方法，但是它们非常重要，所以值得在此重提一遍。）这些是正式的小组活动，目的是把揭露错误的责任从个人转移到小组。由于测试非常耗时，而且错误发现得越晚，代价越高，所以这种活动的目标是在测试开始前发现错误。

使用**走查**的方法，将由一个小组用样本测试输入手动模拟设计或程序，在纸上或黑板上跟踪程序的数据。与全面的程序测试不同，走查并非要模拟所有可能的测试情况，它的目的只是模拟程序员选择的设计或实现程序需求的方法。

在**审查**过程中，将由一位读者（绝对不是程序的作者）逐行读出程序的需求、设计或代码。审查员会预先得到相关资料，而且预期会仔细阅读过这些资料。在审查过程中，审查员会根据审查报告中的记录指出错误之处。他们在预审时已经注释了许多错误。大声朗读的过程只是为了发现更多的错误。与走查一样，小组讨论的主要好处在于讨论是在所有小组成员之间进行的。程序员、测试员和其他小组成员的沟通会在测试开始前发现更多的程序错误。

> **走查**（walk-through）：由一个小组手动地模拟程序或设计的验证方法。
>
> **审查**（inspection）：由团队成员之一逐行读出设计，由其他成员负责指出错误的验证方法。

在高层设计阶段，要拿设计与程序需求进行比较，以确保设计方案包括了所有必需的功能，以及该程序或模块能够与系统中的其他软件正确地连接起来。在低层设计阶段，设计已经具有很多细节，在实现它之前，一定要进行预审。完成编码后，要再审查一次编译过的清单。审查（或走查）可以确保实现与需求和设计一致。成功地完成审查意味着可以开始程序测试了。

走查和审查都要以一种无威胁的方式执行。这些小组活动的重点是去除产品中的瑕疵，而不是设计或代码的作者采用的技术方法。由于这些活动的主持人都不是作者，所以针对的是错误，而不是人。

在过去10年或15年中，Carnegie Mellon大学的软件工程学院在规范大型软件项目的审查过程的研究方面扮演了重要的角色，开办了各种研习班和会议。SEI Software Engineering Process Group (SEPG) Conference上的一篇论文报告了一个项目，该项目采用小组走查和正式审查结合的方式能够把产品的错误减少86.6%。这一过程要应用于生命周期的每个阶段。表17-1展示了在一个维护项目的生命周期的各个阶段发现的每1000行源代码（KSLOC）中的错误数。[6]在维护阶段，50多万行的程序被附加了40 000行源代码。除了测试活动外，每个阶段都要进行正式的审查。

表17-1 维护时发现的错误

阶　段	KSLOC中的错误数
系统设计	2
软件需求	8
设计	12
代码审查	34
测试活动	3

我们刚才讨论的是大型软件项目。在结束这一节之前，我们有必要对“大型”进行一下量化。Space Shuttle Ground Processing System具有50多万行代码；Windows 95具有1000万行代码。大多数大型项目的代码数介于这两者之间。

我们已经指出过，由于大型项目的复杂度，所以要编写没有错误的代码是不可能的。下面是预计错误量的一个参考标准：[7]

- 标准软件：每1000行代码25个bug。
- 好的软件：每1000行代码2个错误。
- Space Shuttle软件：每10 000行代码少于1个错误。

正式验证

如果有工具可以用来定位设计和代码中的错误，而甚至不必运行程序该有多好。虽然听起来不太可能，不过考虑一个来自几何学的比喻。我们不必对每个三角形都证明一次勾股定理，这说明该定理适用于我们用过的每个三角形。我们可以用数学方法证明几何定理，为什么不能这样证明计算机程序呢？

程序正确性的验证独立于数据测试，是计算机科学理论研究的一个重要领域。这项研究的目标是建立证明程序的方法，就像证明几何定理的方法一样。现在已经有证明代码满足规约的必要方法，但是证明通常比程序本身更复杂。因此，验证研究的重点是尝试构建自动化的程序证明器，即验证其他程序的检验程序。

已经有正式的方法可以成功地验证计算机芯片的正确性。一个著名的例子是验证执行实数算术运算的芯片，这项验证获得了英国女王技术成就奖（Queen's Award for Technological Achievement）。牛津大学的程序设计研究组的组长C.A.R.Hoare与MOS Ltd.一起对芯片是否满足规约进行了正式验证。同时执行的还有一种传统的测试方法。《Computing Research News》报道：

“正式的开发方法在两组之间的竞赛中取得了胜利，它只用了大约12个月的时间就完成了，比预计的时间要短。此外，正式的设计指出了许多非正式设计经过几个月的测试而没能指出的错误。最后的设计不仅质量更高，花费更小，完成得也更快。”[8]

硬件层的正式验证技术的成功有望带来软件层验证的成功。但是，软件比硬件复杂得多，所以在不久的将来，不会出现太大的突破。

开源运动[9]

在计算早期，软件（包括它的源代码）是与计算机绑定在一起的。程序员不断地调整和

改编程序，而且很高兴共享他们所做的改进。从20世纪70年代开始，公司开始保留源代码，软件从而成为了一项大生意。

随着Internet的出现，世界各地的程序员几乎无需什么花费就可以进行协作。在Internet上可以得到一个软件产品的简单的版本。程序员仍然对扩展或改进程序充满兴趣。跟踪项目进展的"善意独裁者"掌控着大部分开源项目。如果一种改变或改进获得了同辈开发者们的认可，加入了新的软件版本，那么它一定非常出色。

Linux是最著名的开源项目。Linus Torvolds以UNIX为蓝图，开发了这种操作系统的第一个简单版本，并且一直在观察着它的发展。2001年，IBM花了10亿美金，想把Linux变为一种计算标准。《The Economist》写道：

"有些人对Linux一笑置之，认为它只是个令人陶醉的偶然，不过Linux更像一个即将出现的模式的教科书示例……开源运动是一个大规模的奇迹，全世界已经有上百万的志愿程序员加入了其中，而且还有人在不断地加入，中国和印度的志愿者尤其多。SourceForge是一个开发者的Web站点，现在具有18 000多个开源项目，145 000个程序员在为此忙碌着。"[10]

只有时间才能证明开源软件开发运动是否对制作高质量的软件有所贡献。

Dijkstra对术语"bug"的说明

自从在计算机硬件中发现了蛀虫，计算机的错误就被叫做bug。Edsger Dijkstra却反对我们使用这种术语。他说这种叫法会让人们产生错觉，认为计算机的错误超出了程序员的控制，因为蠕虫可能是在无人看管的情况下偷偷潜入程序的。他认为这是一种智力欺骗，隐藏了程序员自己制造错误的事实。[11]

17.2.3 臭名昭著的软件错误

计算领域中的每个人都有自己喜欢的软件恐怖故事。这里只列出一些小例子。

AT&T停了9小时

1990年1月，AT&T的长途电话网络由于电子交换系统的软件错误中断了9个小时。那天AT&T收到了1.48亿个长途电话和800电话，只有50%被转接了出去。这次故障还引起了数不清的间接破坏：

- 宾馆丢失了预订电话。
- 汽车出租代理丢失了租车电话。
- 美国在线的预订系统通信量降低了2/3。
- 电话推销商估计损失75 000美元。
- MasterCard不能处理200 000个信贷批准。
- AT&T损失了6000万到7500万美元。

正如AT&T的主席Robert Allen所说的，"这是我从商32年来最可怕的噩梦。"[12]

怎么会出现这种情况呢？交换软件的早期版本是能够正确运行的。升级后的系统代码中的软件错误使它对故障交换响应得更快。这个错误发生在一个C代码的break语句中。[13]像Henry Walker在《The Limits of Computing》中指出的，这次崩溃说明了许多软件故障的共同点。在该软件发布之前，它已经经过大量的测试，而且已经正确运行了一个月。除了测试外，开发过程中还进行过代码检阅。一位程序员犯了这个错误，但是其他检阅代码的程序员却没

注意到这个错误。一个相对罕见的事件序列触发了这次故障，这是事先很难预料得到的。而且这个错误出现在为改进一个正确运行的系统而设计的代码中，即出现在维护阶段。E.N.Adams在《IBM Journal of Research and Development》中估计道，在尝试删除大程序中的错误时，约有15%～50%的操作会引入新的错误。

Therac-25

流传最广的软件事故与一台计算机化的放射治疗仪Therac-25有关。在1985年6月到1987年1月之间，Therac-25造成了6次重大的用药过量事故，导致了病人死亡或严重受伤。这些事故据说是应用医疗加速器35年以来最严重的放射事故。

深入分析软件故障已经超出了本书的范围，这里要说明的是，虽然可能的错误只有一种——编码错误，但是深入追究下去，会发现严重的设计失误。Leveson和Turner在《IEEE Computer》发表的文章中加入了下面的评论：

“从Therac-25的故事可以得到的教训是仅仅关注个别的软件bug不能保证系统安全。几乎所有软件在特定条件下都会有意想不到的行为。这里的低级错误是由于缺乏软件工程的经验而构建了依靠软件进行安全操作的机器。此外，软件整体的不安全设计比某个编码错误重要得多。”[14]

政府项目中的bug

1991年2月25日，海湾战争期间，一枚飞毛腿导弹击中了美国陆军的军营，有28名士兵死亡，100多人受伤。由于软件错误，位于沙特阿拉伯Dhahran的美国爱国者导弹发射器没能成功跟踪并阻截伊拉克的飞毛腿导弹。不过这个错误不是编码错误，而是设计错误。其中的一个运算涉及1/10的乘法，这个数在二进制中是无尽的。在100个小时的发射操作中，这种算术错误累积的误差是0.34秒，足够使导弹偏离它的目标。[15]

审计院总结道：

“爱国者从来没有阻击过飞毛腿导弹，而且我们也没有预计它要连续运行这么长时间。在事故发生两周前，陆军官方收到的以色列数据说明在系统连续运行了8小时后，已经出现了误差。于是陆军官方修改了软件，以提高系统的精确性。但是，直到2月26日，飞毛腿导弹事件发生后的第二天，修改好的软件才到达Dhahran。”[16]

Gemini V的着陆地点距预计的点100英里。什么原因？是导航系统的设计没有将地球围绕太阳的转动考虑在内。[17]

1999年10月，美国发射的火星气候轨道探测器（Mars Climate Orbiter）进入了火星大气层，进入点比预计的低100公里，导致飞船烧掉了。火星气候轨道探测器任务失败调查小组的主席Arthur Stephenson总结道：

“导致太空船销毁的根本原因是一个地面导航软件没能像NASA宣布的那样把英语转换成度量单位……失败调查小组还发现了其他导致错误的重要因素，它们使错误拖延下来，结果使飞船进入火星的路径出现了很大的误差。”[18]

1962年7月美国发射的水手1号（Mariner 1）金星探测器几乎一发射就转变了航向，所以不得不被销毁了。这个问题是由下面这行Fortran代码引起的：

```
DO 5 K=1.3
```

其中的句号应该是个逗号。由于这个输入错误，价值1850万美元的太空探索飞船就这么被毁

掉了。

软件错误并不只是美国政府才会犯。1996年6月4日，欧洲空间局发射的无人火箭Ariane 5在升空40秒后就爆炸了。这架火箭开发了十几年，开发费是70亿美元。火箭本身和它携带的货物价值5亿美元。究竟发生了什么问题呢？一个相对于平台的水平速率是64位的浮点数，大于32 767，结果被转换成了16位的整数，导致火箭转变了航迹，然后解体，爆炸。

17.3 问题

生活中总是充满了各种问题。对有些问题，能够轻松地开发和实现计算机解决方案。对有些问题能实现计算机解决方案，但不能得到日常生活中的结果。有些问题在具有足够的计算机资源的情况下能够开发和实现计算机解决方案。有些额外问题可以证明是没有解决方案的。在介绍这些问题分类之前，必须先介绍一下比较算法的方法。

17.3.1 算法比较

前面的章节中介绍过，大部分问题的解决方案不止一种。如果你询问去Joe's Diner的路（请参阅图17-2），可能会得到两种等价的正确答案：

1. “走高速公路，到Y'all Come Inn之后，左转。”
2. “走winding country road，到Honeysuckle Lodge之后，右转。”

虽然这两种答案不同，但无论走哪条路，都可以到达Joe's Diner，所以这两个答案都是正确的。

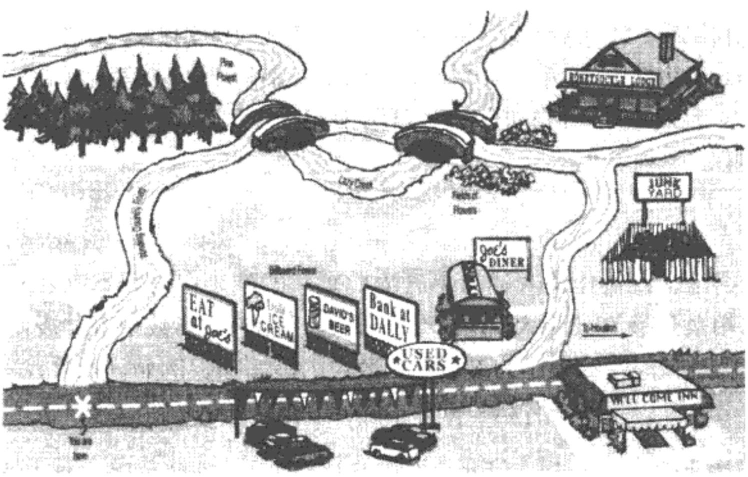


图17-2　同一个问题的等价有效解决方案

如果问路的请求中包括特殊要求，那么一种解决方案可能比另一种好。例如，“我要迟到了，哪条路到Joe's Diner最快？”这时要用第一种方案。如果要求是“有没有安静的小路可以到Joe's Diner？”就要用第二种方案。如果没有特殊要求，那么可以根据个人喜好进行选择，你喜欢哪条路？

关于算法的选择通常是由效率决定的。哪个算法花费的计算时间最少？哪个算法完成作业的工作量最小？这里我们指的是计算机所做的工作量。

要比较两个算法的工作量，首先要定义一组客观的度量标准。算法分析是理论计算机科

学的一个重要研究领域，在高级计算课程中，你可以看到该领域中的大量工作。这里我们只介绍这个主题的一小部分，让你能够比较两个任务相同的算法，理解算法的复杂度构成了一个从易于解决到不能解决的连续统。

程序员如何衡量两个算法执行的工作呢？首先想到的是对算法编码，然后对比两个程序的运行时间。执行时间较短的算法显然是比较好的算法。是这样吗？使用这种方法，只能确定程序A在特定的计算机上比程序B有效。执行时间是特定计算机特有的。当然，可以在所有可能的计算机上测试算法，但我们需要一个更通用的方法。

第二种方法是计算执行的指令数或语句数。但是，使用的程序设计语言不同，以及程序员的个人风格不同，都会对这种衡量方法有影响。为了标准化这种衡量方法，可以计算算法中执行关键的循环的次数。如果每次迭代的工作量相同，那么这种方法就给我们提供了算法效率的有效衡量标准。

另一种方法是把算法中的一个特定基本操作分离出来，计算这个操作执行的次数。例如，假设要求一个整数列表中的元素的和。要衡量所需的工作量，就要计算整数加法操作的次数。对于有100个元素的列表，需要99次加法运算。但要注意，并非真的要去计算加法运算的次数，它是列表中的元素个数（N）的函数。因此，可以用N表示加法运算的次数，对于有N个元素的列表，需要$N-1$次加法运算。现在可以比较一般情况的算法性能，而不必只是比较特定列表大小的情况了。

标签

有些纺织品制造商伪造自己产品的原产地，以便在衣服进口到美国时逃税。新的标记系统可以把信息编码成肉眼看不到，这样每年可以减少几百万美元的税收损失。这种系统用近红外扫描仪可以读取的微标记给纺织品作标记。扫描仪可以识别纺织品的产地、类型、状态和成分。即使在粗糙的制造过程中，包括冲刷、漂白和染色，这种微标记也能保存下来。这种技术还可以用于国防、存货跟踪和控制以及军事应用。

大O分析

我们已经介绍过，以操作输入的大小（如要求和的列表中的元素个数）的函数来衡量工作量。我们可以用数量级（或叫**大O**，这是字母O，不是0）的数学符号表示这个函数的近似值。函数的数量级是以问题的大小为参数的函数中的最高项。例如，如果

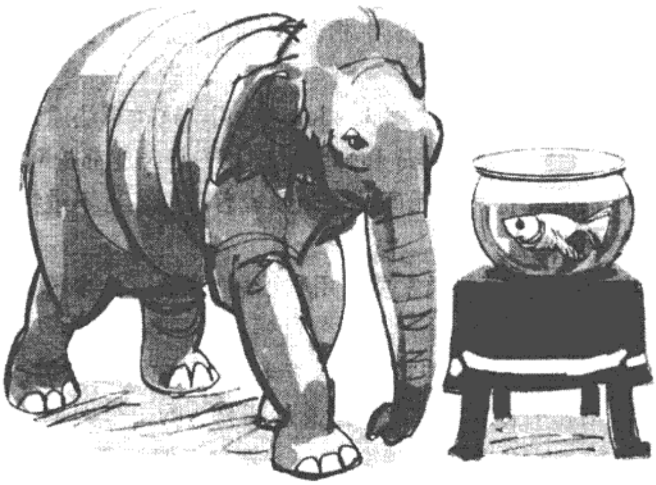
$$f(N) = N^4 + 100N^2 + 10N + 50$$

那么$f(N)$ 的数量级是N^4，用大O符号表示就是$O(N^4)$。也就是说，对于较大的N，N^4在函数中占支配地位。$100N^2 + 10N + 50$并非不重要，只是随着N越来越大，其他的因素就会变得无足轻重，因为N^4支配着这个函数的量级。

大O符号（Big-O notation）：以函数中随着问题的大小增长得最快的项来表示计算时间（复杂度）的符号。

为什么可以舍弃低数量级的项呢？举例来说，如果我们想买大象和金鱼，考虑两家宠物销售商，我们只需要对比大象的价格，金鱼的价格根本微不足道。在算法分析中，随着问题大小增长得最快的项支配着整个函数，把其他项明显地降到了“噪音”的水平。大象太大，

以至于我们可以忽略金鱼。同样地，对于较大的N，N^4比50、$10N$，甚至$100N^2$都大得多，以至于可以忽略这些项。这并不意味着这些项对计算时间没有影响，只是说它们在N比较“大”时对我们的估计没有显著影响。



什么是N？N表示问题的大小。大多数问题涉及第9章讨论过的数据结构。每种结构由元素构成。我们要开发算法，把元素添加到结构中，以及修改元素，或把元素从结构中删除。用N可以描述这些操作的工作量，N是结构中的元素个数。

假设要把一个列表中的所有元素写入一个文件。工作量有多大？答案是由列表中的元素个数决定的。算法如下：

```
Open the file
While more elements in list
  Write the next element
```

如果N是列表中的元素个数，那么要实现这个任务需要的时间是（$N\times$写入一个元素的时间）＋打开文件的时间。

这个算法的时间复杂度是$O(N)$，因为执行任务所需的时间与元素个数N成比例（外加一点打开文件的时间）。在决定大O的近似值时，为什么能忽略打开文件的时间呢？假设打开文件必需的时间是一个常量，那么算法的这个部分就相当于金鱼。如果列表中只有几个元素，打开文件的时间可能会看来很重要，但对于较大的N，写入元素的操作与打开文件比起来就像大象。

算法的数量级并没有表明解决方案在我们的计算机上运行需要花费多少微秒。有时，我们需要这种信息。例如，一个字处理器的要求写到，该程序必须能在（特定计算机上）120秒以内对50页文档进行拼写检查。对于这种信息，就不能使用大O分析，而需要其他的衡量方法。我们可以对一种数据结构的不同实现进行编码，然后运行测试，记录运行前和运行后的计算机时钟上的时间。这种基准测试可以告诉我们，这些操作在特定的计算机上用特定的编译器执行需要花费多少时间。但大O分析无需引用这些因素就可以比较算法。

常见的数量级

$O(1)$ 叫做有界时间，即工作量是个常数，不受问题大小的影响。给具有N个元素的数组中的第i个元素赋值，复杂度是$O(1)$，因为可以通过索引直接访问数组中的元素。虽然有界时

间通常又叫做固定时间，但工作量却不必一定是固定的，它只是有一个常量界限而已。

$O(\log_2 N)$ 叫做对数时间，即工作量是问题大小的对数。每次都把问题的数据量减少一半的算法通常都属于这个类别。用二分检索法在有序列表中查找一个值，复杂度是$O(\log_2 N)$。

$O(N)$ 叫做线性时间，即工作量是一个常数乘以问题的大小。输出具有N个元素的列表中的所有元素，复杂度是$O(N)$。在无序列表中检索一个值的复杂度也是$O(N)$，因为必须检索列表中的每一个元素。

$O(N \log_2 N)$（由于缺乏更好的术语）叫做$N \log_2 N$时间。这类算法通常要应用N次对数算法。比较好的排序算法（如快速排序、堆排序和合并排序）的复杂度都是$N \log_2 N$。也就是说，这些算法能用$O(N \log_2 N)$ 的时间把一个无序列表转换成有序列表，不过快速排序算法对于某些输入数据的时间复杂度是$O(N^2)$。

$O(N^2)$ 叫做二次时间。这类算法通常要应用N次线性算法。大多数简单排序算法的时间复杂度都是$O(N^2)$。

$O(2^N)$ 叫做指数时间。这类算法非常耗时。在表17－2中可以看到，随着N的增长，指数时间增长得非常快。国王和稻谷的故事就是指数时间算法的一个例子，在这个故事中，问题的大小就是稻谷的颗粒数。（还要注意的是，最后一列的值增长得非常快，以至于这个量级的问题所需的计算时间超出了预计的宇宙生命期限！）

表17-2 增长率的对比

N	$\log_2 N$	$N \log_2 N$	N^2	N^3	2^N
1	0	1	1	1	2
2	1	2	4	8	4
4	2	8	16	64	16
8	3	24	64	512	256
16	4	64	256	4096	65 536
32	5	160	1024	32 768	4 294 967 296
64	6	384	4096	262 144	在超级计算机上约为5年
128	7	896	16 384	2 097 152	以纳秒计约为宇宙年龄的600 000倍（估计为60亿年）
256	8	2 048	65 536	16 777 216	不要问这个问题

$O(n!)$ 叫做阶乘时间。这类算法甚至比指数时间的算法更耗时。货郎担这个图论问题（见17.3.4节）就是一个阶乘时间算法。

数量级是问题大小的多项式的算法叫做**多项式时间算法**。第2章介绍过，多项式是两个或多个代数项的和，每个代数项是一个常量乘以一个或多个变量的非负整数次幂。因此，多项式算法就是数量级能够用问题大小的幂表示的算法，算法的大O符号是多项式中的最高次幂。所有的多项式时间算法都被定义为**P类**算法。

> **多项式时间算法**（polynomial-time algorithms）：复杂度能用问题大小的多项式表示的算法。
> **P类**（class P）：由所有多项式时间算法构成的类。

把常见的复杂度量级看作一个个箱子，我们可以以此对算法复杂度排序（如图17-3所示）。对于较小的问题，一个箱子中的算法可能真的比下一个更有效的箱子中的等价算法快。随着

问题增大，不同箱子中的算法之间的差别会随之增加。在选择同一个箱子中的算法时，就不会再忽略金鱼了。

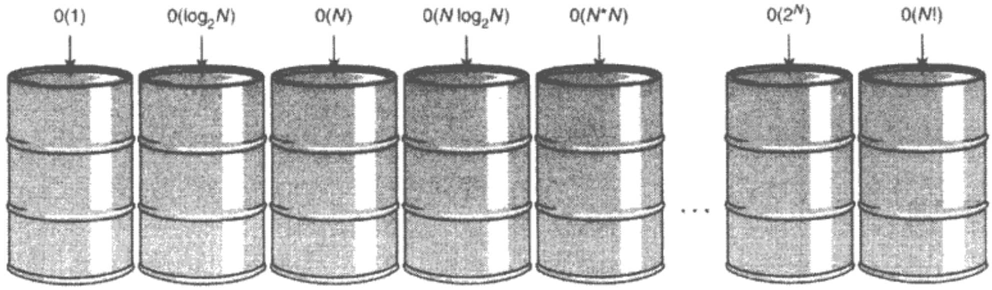


图17-3 复杂度的顺序

家庭洗衣量：一个比喻

每周一个家庭要花多少时间洗衣服？可以用下面的函数来回答：

$$f(N) = c*N$$

N表示家庭成员数，c是洗每个人的衣服需要花费的时间。这个函数的复杂度是O(N)，因为整体的洗衣时间是由家庭成员数决定的。对于不同的家庭，常量c可能稍有不同，这是由洗衣机的容量和他们折叠衣服的速度决定的。也就是说，两个家庭的洗衣时间可以用下面的两个函数表示：

$$f(N) = 100*N$$
$$g(N) = 90*N$$

现在，如果爷爷和奶奶来第一个家庭住一到两个星期会出现什么情况？洗衣时间的函数将变为：

$$f(N) = 100*(N+2)$$

我们仍然说这个函数的复杂度是O(N)。为什么？多出了两个人，漂洗、烘干和叠衣服的时间不会增加吗？当然会增加。如果N很小（这个家庭只有妈妈、爸爸和孩子），那么增加两个人需要多花的洗衣时间还是很明显的。不过随着N的增加（这个家庭有妈妈、爸爸、12个孩子和一个保姆），多两个人区别就不大了。（这家的洗衣时间就像大象，而客人的洗衣时间就像金鱼。）当用大O比较算法时，我们关心的是N比较大的情况。

如果我们的问题是“我们能及时洗完衣服，赶上7:05的火车吗？”那么我们想要的是精确的答案。大O不能给我们这些信息，它给的是个近似值。因此，如果100*N、90*N和100*(N + 2) 的复杂度都是O(N)，那么我们如何分辨哪个更好呢？用大O符号，我们不能回答哪个更好，对于较大的N，它们基本上是等价的。我们能找到更好的洗衣算法吗？如果这个家庭中了彩票，那么他们就可以在距离他们家15分钟车程（往返约为30分钟）的专业洗衣店洗衣。现在，这个函数是：

$$f(N) = 30$$

这个函数的复杂度是O(1)。这个答案独立于家庭成员数。如果车程变为5分钟，那么

该函数就变为：

$$f(N) = 10$$

这个函数的复杂度仍然是O(1)。采用大O进行比较，这两种专业洗衣店的解决方案是等价的，无论有多少位家庭成员，也无论有多少客人，这个家庭用来洗衣的时间都是个常量。(我们不关心专业洗衣店洗衣的时间。)

17.3.2 图灵机

这本书已经不止一次提到过Alan Turing这个名字。是他，在20世纪30年代开发了计算机器的概念。他的兴趣并非实现这台机器，而是用它作为一种模型，来研究计算的限度。

这种著名的模型就是图灵机，由具有读写头的控制部件构成，能够在无限的带子上读写符号，带子被分成了单元。这个模型的基础是一个人用铅笔和橡皮在长长的纸带上进行简单的运算。纸上的每一行（一个单元）包含一个有限字符集中的符号。从第一行开始，这个人分析其中的符号，或者保留它，或者用字符集中的另一个符号替换它。然后他移到下一行，重复上述操作。

Alan Turing



《时代》杂志把Alan Turing选为20世纪最具影响力的100位名人之一。Turing的传记如下：

这位古怪的剑桥年轻导师所做的就是构造一台假想机，这是一台类似于打字机的简单装置，能够扫描或读取一条理论上无限长的带子上的指令。这台扫描器从带子上的一个方格移到下一个方格，响应序列的指令，并修改它的机械响应，Turing证明了这种过程的输出可以复制人类的逻辑思维。

这种假想的装置很快就有了一个名字——图灵机，Turing的另一个构想也是如此。由于机器的行为是由带子上的指令控制的，改变这些指令，就可以使这台机器执行各种功能。换句话说，采用不同的带子，同一台机器既可以执行运算，又可以下棋，还可以执行其他所有具有计算性的任务。因此，他的装置得到了一个新的、更显要的名字——通用图灵机。

……

现代计算机的诞生汇集了无数的构想和高级技术，很难把它的发明归功于某个人。不过，每个在敲打键盘的人、在打开电子制表软件或字处理程序的人，都在使用具体化的图灵机。[19]

Alan Turing生于1912年6月，他的父亲Julius Mathison Turing是印度行政参事会（Indian Civil Service）的成员，母亲Ethel Sara Stoney是Madras铁路的首席工程师的女儿。他的父亲和母亲大部分时间都是在印度度过，而他的哥哥则是在英国很多抚养孤儿的家庭中长大的，这种情况一直持续到1926年他们的父亲退休。

当时的英国公立学校系统不能培养创新思想，所以Turing适应不了这种教学方式。他的书法受到批评，英语学得很吃力，甚至数学都不能给出预期的常规答案。他13岁进入

了Sherborne School，这里的校长说，如果他是当科学家的材料，那么在公立学校只会浪费时间。但是，他的母亲非常看重公立学校的教育，所以他只好坚持下去。这一时期他的精神支柱有两个，一个是他的自学能力，另一个是他和Christopher Morcom的友谊，后者是比他高一年级的学生。Morcom是他重要的精神支柱，但这种支持在两年后由于Morcom的猝死而消失了。

1931年，Turing进入了剑桥大学国王学院，开始研究数学。国王学院的学习氛围鼓励自由思想，这使他第一次找到了精神家园。1934年毕业后，他于1935年凭一篇论文"On the Gaussian Error Function"（证明了概率论的基本结果）被推举为国王学院的研究员。

从此，Turing依据他跟Max Newman学过的数学基础课程，开始了可判定问题的研究。在1936年发表的论文中，Turing引入了图灵机的概念。这些概念的背景是是否存在明确的方法或过程，能够确定任何指定的数学断言是否是可证明的。与此同时，Alonzo Church在普林斯顿从事同一主题的研究已经小有名气了，所以Turing的论文一直推迟到他能够引用Church的工作才发表。此后，Turing在普林斯顿作为学生，与Church和von Neumann一起工作了两年。

二战爆发时，Turing开始为政府效力。美国《时代》杂志的报道如下：

根据Turing发表的论文，他被分配到位于Buckinghamshire Bletchley Park的Government Code and Cypher School工作。正式的系统是要破解纳粹在总部和军队通信时使用的密码，只要能对正式系统中可能的排列组合有所贡献的人都能参与这项工作，其中包括数学家、国际象棋冠军、埃及古物学者。限于保密规定，Turing在此的角色直到他去世后很久才公诸于世。与计算机的发明一样，Bletchley Park的工作也是集体智慧的结晶。不过，Turing在类似计算机的机器原型设计方面仍然扮演着至关重要的角色，这台机器能够高速破解纳粹发给北大西洋上的U-潜艇的密码。[20]

由于Turing在战争中的贡献，他于1945年被授予了英国皇家勋章。

他在伦敦的国家物理实验室尝试过构建一台计算机，失败之后，他回到了剑桥，继续自己的工作和写作。战争时期的团结协作精神盖过了官僚作风，但是战争结束后，这种精神就逐渐消散了，所以ACE（自动计算引擎）再也不能构建成了。1948年，Turing成了曼彻斯特大学的计算实验室的副主管。这个暧昧的头衔毫无意义，接下来的几年Turing从事了多个主题的研究。

1950年，他发表的一篇论文反映了他的主要兴趣之一，即机器可以思考吗？著名的图灵测试就是出自这篇文章。此外，他还对形态发生（即开发活有机体中的模式和组成）感兴趣。而且他对可判定问题和量子论的研究一直没有停止过。

1954年6月7日，Turing死于氰化物中毒，在他的床边发现了半个咬过的苹果。他的母亲相信他是在进行实验时猝死的，而验尸官则判断他是自杀的。几年前，伦敦的West End和百老汇上演过一部独角戏，叫做《Breaking the Code》，这部获奖作品使观众对Turing这个才华横溢且复杂的人物有了简单的了解。

图灵机的控制部件模拟了这个人。人的决策过程由控制部件能执行的一系列指令表示。每个指令可以：

- 从带子上的一个单元读取一个符号。

• 把一个符号写入带子上的一个单元。

• 使带子向左移动一个单元，或向右移动一个单元，或者保持不动。

如果我们允许一个人自己替换符号，这些动作其实是模拟了一个使用铅笔的人。如图17-4所示。



图17-4　图灵机的处理

为什么这样一个简单的机器（模型）这么重要呢？一个广为接受的说法是*任何能直观计算的问题都能被图灵机计算*。这个说法叫做Church-Turing理论，是以Turing和Alonzo Church的名字命名的，后者是开发了另一个类似的模型λ演算的数学家，是Turing在普林斯顿的同事。计算机科学的理论课程会深入地介绍Turing和Church的工作。

从Church-Turing理论我们可以得出这样的结论，如果证明了一个问题的图灵机解决方案不存在，那么这个问题就是不可解决的。我们将在下一节介绍这个问题。

17.3.3　停机问题

计算（程序）终止并不总是很明显的。第6章介绍过重复一个过程的概念，第8章介绍过不同的循环类型。有些循环会明显地终止，而有的则不会（无限循环），还有些循环是根据输入的数据或循环中发生的计算终止的。在一个程序运行的过程中，很难分辨它是进入了无限循环还是需要更多的时间来运行。

因此，如果可以预言一个具有特定输入的程序不会落入无限循环，是非常有用的。**停机问题**以下面的方式重新阐述了这个问题，*即给定一个程序和它的输入，确定该程序采用这样的输入最终是否能停止。*

> **停机问题**（halting problem）：确定对于指定的输入一个程序最终是否能停止的问题，是不可解决的。

最明显的方法是用特定的输入运行程序，看会发生什么情况。如果它停止了，答案显而易见。如果它不停止呢？一个程序要运行多久你才会判定它落入了无限循环？显然，这种方法有问题。遗憾的是，其他的方法也都有问题。这个问题是不可解决的。这个断言的证明是“没有图灵机程序可以确定一个程序是否在指定的输入下会停止。”

那么如何证明一个问题是不可解决的，或者只是我们还没找到解决方案而已呢？可以尝试每种提出的解决方案，证明每种方法都有问题。由于已知的解决方案可能很多，而且还有很多是未知的，所以这种方法看来行不通。然而这种方法构成了图灵解决方案的基础。在他的证明中，就是从提出的解决方案入手，然后证明它们是行不通的。

假设存在一个图灵程序SolvesHaltingProblem，对于任何程序Example和输入SampleData，它都能确定Example采用SampleData是否会停止。也就是说，程序SolvesHaltingProblem以程序Example和输入SampleData作为参数，如果Example能停止，则输出“Halts”，如果Example具有无限循环，就输出“Loops”。图17-5展示了这种情况。

还记得吗？在计算机中，程序（指令）和数据是相似的，都是位组合。程序和数据的区别在于控制部件如何解释位组合。因此，如果把Example自身作为SampleData，那么SolvesHaltingProblem就要以Example程序和它的副本作为参数，来判断Example以其自身作为

输入是否会停止。如图17-6所示。

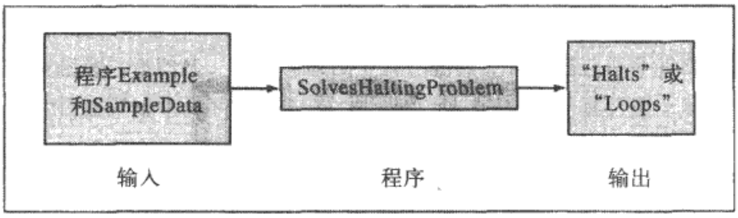


图17-5 为解决停机问题提出的程序

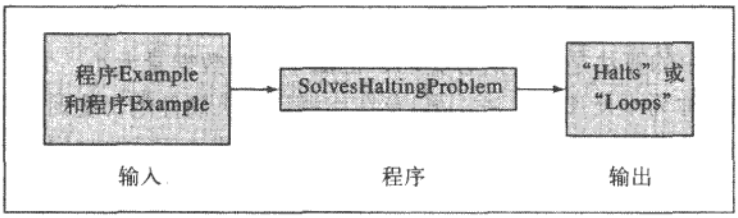


图17-6 为解决停机问题提出的程序

现在我们构造一个新程序NewProgram，以Example作为程序和输入数据，采用SolvesHaltingProblem的算法，如果Example会停止，就输出"Halts"，如果Example具有无限循环，就输出"Loops"。如果输出了"Halts"，NewProgram将创建一个无限循环；如果输出的是"Loops"，NewProgram将输出"Halts"。图17-7展示了这种情况。

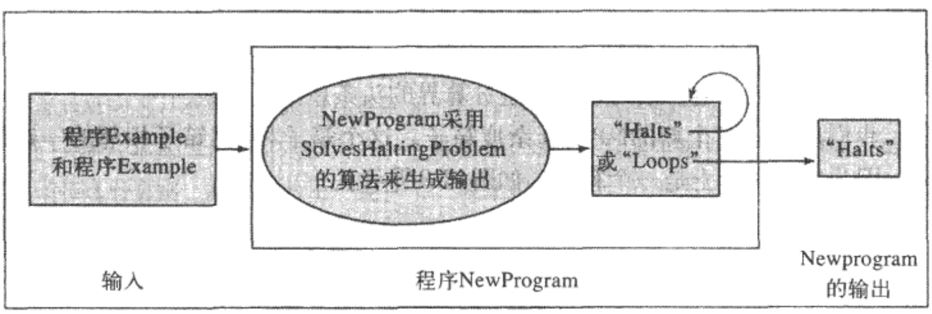


图17-7 NewProgram的构造

看明白这个证明了吗？把SolvesHaltingProblem应用到NewProgram上，以NewProgram作为输入数据。如果SolvesHaltingProblem输出"Halts"，那么NewProgram就落入了无限循环。如果SolvesHaltingProblem输出"Loops"，NewProgram将输出"Halts"并停止。无论哪种情况，SolvesHaltingProblem所给答案都是错的。由于SolvesHaltingProblem至少会对一种情况给出错误答案，所以它不适用于所有情况。因此，任何提出的解决方案都是有问题的。

17.3.4 算法分类

图17-3用箱子表示常见的数量级。现在我们知道，最右边还有一个箱子，存放的是不能解决的算法。让我们来重组这些箱子，把所有多项式算法放在P类箱子中，把指数和阶乘算法放在一个箱子中，再加一个不能解决的算法的箱子。如图17-8所示。

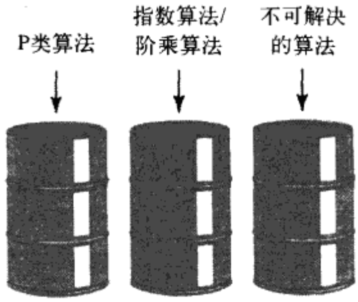


图17-8 重组的算法分类

虽然中间箱子中的算法是有解决方案的，但由于无论数据量大小，它们都要执行很长的时间，所以它们被称为*难处理的算法*。第1章在回顾计算机硬件的历史时提到过并行计算机。如果同时使用足够多的处理器，某些问题能在合理的时间（多项式时间）内解决吗？是的，可以。如果使用大量的处理器，就能在多项式时间内解决的问题叫做**NP类问题**。

显然，P类问题也是NP类问题。理论计算学中的一个未决问题是，只有用多个处理器才能解决的NP类问题是否也是P类问题。也就是说，这些问题是否存在多项式算法，而我们还没发现（发明）。我们不知道这个问题的答案，不过计算机科学的理论研究者一直在忙于寻求这些问题的解决方案。解决方案？是的，判断P类是否等价于NP类的问题已经被简化为找到其中一个算法的解决方案。有一类特殊的问题叫做**NP完全问题**。这些问题属于NP类，它们的属性可以互相映射。如果找到了这个类中的一个算法的单处理器多项式解决方案，那么所有算法都会有这样的解决方案，因为一个解决方案可以映射到其他所有问题的解决方案。如何映射以及为什么能这样映射已经超出了本书的范围。但是，一旦其中某个算法的解决方案被发现了，你一定会立刻知道，因为这一定是计算界的头条新闻。

现在，我们应该多了个新的NP类复杂度箱子。这个箱子和P类箱子相邻的一边用虚线标示了出来，因为它们实际上是一个箱子。如图17-9所示。

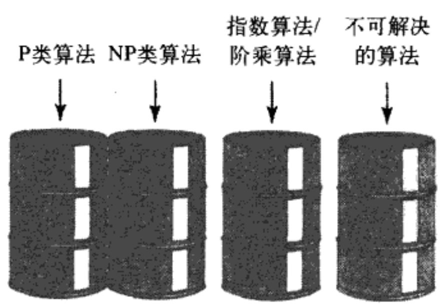


图17-9 加入了NP类

P类问题（class P problems）：用一个处理器能在多项式时间内解决的问题。

NP类问题（class NP problems）：用足够多个处理器能在多项式时间内解决的问题。

NP完全的问题（NP-complete problems）：NP类问题的子集，如果发现了其中任何一个问题的单处理器多项式时间的解决方案，那么其他所有问题都存在这样的解决方案。

货郎担问题

一个经典的NP问题叫做货郎担问题。一个货郎要走访他的销售区内的所有城市。为了有效地走访每个城市，他想找到一条路线，在返回起点之前，要经过且只经过每个城市一次。可以用图的顶点表示城市，图的边表示城市间的路。每条边上标有城市之间的距离。这个解决方案成了著名的图论算法，它的单处理器解决方案的复杂度是O(*N*!)。

小结

硬件、软件和要解决的问题自身都对计算机的问题求解有限制。数字本身是无限的，而计算机能表示的数字却是有限的。这种限制会导致算术运算错误，生成不正确的结果。硬件部件则会磨损，计算机之间或计算机内部的数据传输则会造成信息损失。

大型软件项目的大小和复杂度几乎一定会导致产生错误。虽然测试可以证明有错误，但却不能证明没有错误了。构建好的软件的最佳方法是从项目一开始就关注它的质量，应用软件工程的规则。

从非常易于解决的，到根本不能解决的，问题的种类很多。使用大O分析可以根据由问题大小决定的增长速率来对比算法。多项式时间算法是大O复杂度能够用问题大小的多项式表示的算法。P类问题是能用单处理器在多项式时间内解决的问题。NP类问题是能用足够多的处理器在多项式时间内解决的问题。Turing证明过，停机问题是没有解决方案的。

道德问题：深度链接

WWW对社会的强烈冲击毫无疑问归功于它简化了通信和信息交换的能力。在这种革命性的媒体中，人们几乎可以即时交流、进行研究或发布自己的思想。用户只要跟着超链接，就可以轻松地从一个网页冲浪到另一个网页，超链接会把他们带到相关的或感兴趣的主题。超链接可以是文本或图像，它们会响应鼠标点击，发送给用户一个新页面，这个页面通常来自用户所在的网站之外的网站。通过连接这些页面，超链接为用户提供了一项重要的服务，这也是Web的一个定义特性。在网络发展的早期，链接是网络的要素，被看作电脑空间不可或缺的向导。但是，随着网络发展日益成熟，深度链接引起了争议。所谓深度链接，就是一个网页中包含的超链接引用了深藏在另一个网站中的网页，而不是这个网站的主页。虽然许多公司欢迎无意中发现了他们网页（无论是不是他们的主页）的访客，但有些公司则觉得深度链接是不合理的，这种技术不适当地绕过了他们站点的“门户”。

1997年，Ticketmaster.com公司控告Microsoft不适当地链接到了它的网站，使这个问题得到了公众的关注。Microsoft公司的城市导购站点“Sidewalk”提供了Ticketmaster.com上某些事件的票务信息的链接。暂且不提这些链接造成的网络拥堵，Ticketmaster.com公司认为它应该能够控制其他链接如何链接到自己的站点，而深度链接不适当地绕过了它们的广告。虽然这个案子得到了庭外和解，但不久之后Ticketmaster.com公司又对它的竞争对手提出了同样的控诉。在Tickets.com面临的众多控诉中，就包括不适当的链接。Ticketmaster.com公司声称Tickets.com公司直接链接到了它的内部网页而不是主页，这是不正当的商业行为。Ticketmaster.com公司列出了许多控诉，其中就有深度链接损害了其广告收益。法庭裁定Tickets.com没有违反版权法，因为它不是复制所链接的网页的内容，用新的形式发布出来，而且两个站点之间的关系也不太可能被误解。但这个裁决并不意味着深度链接的问题已经解决了。其他公司（如eBay和Universal Studios）也有类似的举动，

以防止对其站点的深度链接。在Internet规则的开发和具体化的过程中，这个问题将不断引发争论。

练习

为练习1～15中的定义或应用找到匹配的大O符号。

A. O(1) B. $O(\log_2 N)$

C. $O(N)$ D. $O(N \log_2 N)$

E. $O(N^2)$ F. $O(2^N)$

G. $O(N!)$

1. 阶乘时间。
2. $N \log N$时间。
3. 线性时间。
4. 二次时间。
5. 指数时间。
6. 对数时间。
7. 有界时间。
8. 与问题大小无关的时间。
9. 每一步都能把数据量减少一半的算法。
10. 合并排序和堆排序。
11. 选择排序和冒泡排序。
12. 添加一个具有N个数字的列。
13. 国王与稻谷的故事证明的复杂度。
14. 货郎担问题。
15. 如果数据是有序的，快速排序会退化到什么复杂度。

为练习16～20中的算法找到匹配的技术名称。

A. 偶数奇偶校验 B. 奇数奇偶校验

C. 校验数位 D. 误差校正码

E. 校验位

16. 硬件中每个字节的附加位，确保每个字节中都有偶数或奇数个1。
17. 极端的冗余，对每个值都保留两个副本。
18. 字节加校验位中的1的个数为奇数。
19. 字节加校验位中的1的个数为偶数。
20. 求数字中每个数位的和，然后把和的个位数与数字存储在一起的模式。

判断练习21～30中的陈述的对错：

A. 对 B. 错

21. （1＋x－1）一定等于x。
22. 表示误差就是舍入误差。
23. 软件验证活动仅限于实现阶段。
24. 软件项目中的一半错误都发生在设计阶段。
25. 大多数大型软件项目都是由一个天才人物设计，然后交给程序员组开发的。
26. 在软件生命周期中，错误发现得越晚，修正它的代价越小。
27. 程序的正式验证只停留在理论研究阶段，至今还没有实行过。
28. 大O符号可以告诉我们一个解决方案运行了多少微秒。
29. 软件工程是计算学的一个分支，出现于20世纪60年代。
30. 现有软件的维护和升级已经变得比构建新系统更重要。

练习31～61是问答题或简答题。

31. 请定义表示误差、化零误差、下溢和溢出。讨论这些术语的相关性。
32. 请说明采用下列字长能表示的整数范围。
    a) 8位 b) 16位
    c) 24位 d) 32位
    e) 64位
33. 当发生下溢时，还能采取符合逻辑的动作补救，但是发生溢出却没有补救措施，请解释为什么。
34. a) 说明如何用每个节点存放一个数位的链表表示数字1066和1492。
    b) 用链表表示这两个数的和。
    c) 列出一个算法，说明如何用计算机执行上述运算。
35. 请解释Titanic效应与硬件故障的关系。
36. 你遇到过哪些硬件故障？请解释。
37. 给定下列8位代码，如果采用奇数奇偶校验，那么它们的校验位是什么？
    a) 11100010 b) 10101010
    c) 11111111 d) 00000000
    e) 11101111

38. 给定下列8位代码，如果采用偶数奇偶校验，那么它们的校验位是什么？
    a) 11100010　　b) 10101010
    c) 1111111　　d) 00000000
    e) 1101111
39. 给定下列数字，它们的校验数位是什么？
    a) 1066　　b) 1498
    c) 1668　　d) 2001
    e) 4040
40. 使用练习39中的校验数位可以检测到什么错误？
41. 给定下列数字，如果同时用偶数数位的和的个位数和校验数位进行验证，额外的数位是什么？
    a) 1066　　b) 1498
    c) 1668　　d) 2001
    e) 4040
42. 给定下列数字，如果同时用奇数数位的和的个位数和校验数位进行验证，额外的数位是什么？
    a) 1066　　b) 1498
    c) 1668　　d) 2001
    e) 4040
43. 练习41和42中的表示法如何改进了简单的校验数位检测错误的功能？
44. 请解释软件生命周期的概念。
45. 在软件项目中，错误多发于哪些地方？
46. 为什么错误发现得越晚，修正错误的花费越高？
47. 请对比走查和审查。
48. 为什么一个程序可能被证明为正确的但却仍然是无价值的呢？
49. 请列举至少5处可能发生软件错误的地方。
50. AT&T的软件错误有哪些典型之处？
51. 什么是正式验证？
52. 请解释大象和金鱼的比喻。
53. 请定义多项式时间。
54. 为什么多项式时间算法的大O符号除了最高项外可以舍弃多项式中的其他项？
55. 给出下列多项式的大O符号。
    a) $4x^3 + 32x^2 + 2x + 1003$　　b) $x^5 + x$
    c) $x^2 + 124578$　　d) $x + 1$
56. 请解释复杂度的箱子这个比喻。
57. 谁制造了图灵机？
58. 图灵机如何模拟一个具有纸和笔的人？
59. 是否存在没有解决方案的问题？
60. 请说明停机问题。
61. 数据和程序在计算机中是相似的，如何利用这一事实证明停机问题是不可解决的？

思考题

1. 在Web上检索有关Pentium芯片错误的信息。尝试不同的关键字和关键字组合，记录每次检索得到的信息数量。从中选取至少3篇文章，用你自己的话描述这个问题。
2. 在Web上检索下列问题的答案。
    a) Russian Phobos 1太空船是自行毁灭的吗？
    b) 什么原因推迟了Denver机场的开业时间？
    c) 修复英国伦敦的救护车调度系统故障的花费是多少？
    d) 1998年，美国军舰Yorktown沉入水中几个小时。是什么软件错误造成了这次事故？
3. 一位教授在地方的士兵俱乐部中作了一次关于计算限制的讲座。一位听众说“但我认为根本不存在任何限制。”如果你是这位教授，你会如何回答他？
4. 你如何看待深度链接？访问另一个站点一定要经过这个站点的主页吗？如果某人访问了你的站点内部的某个页面，你会觉得不舒服吗？这样的行为是不是如同断章取义？
5. 许多商业站点都是通过广告赚钱。那么通过深度链接绕过广告道德吗？是否应该通过一项法律禁用深度链接呢？
6. 如果你有自己的站点，那么你有链接吗？你的链接都是深度链接吗？读完有关深度链接带来的问题的文章后，你准备修改这些链接吗？

参考文献

第1章

1. G. A. Miller, "Reprint of the Magical Number Seven Plus or Minus Two: Some Limits on Our Capacity for Processing Information," *Psychological Review* 101, no. 2 (1994): 343–352.
2. "Beyond All Dreams" <http://www.mith.umd.edu/flare/lovelace/index.html>
3. Written by Chip Weems, adapted from: Nell Dale, Chip Weems, and Mark Headington, *Java and Software Design* (Sudbury, MA: Jones and Bartlett Publishers, Inc., 2001): 352–3.
4. P. E. Grogono and S. H. Nelson, *Problem Solving and Computer Programming* (Reading, Mass: Addison-Wesley, 1982): 92.
5. P.E. Cerruzzi, *A History of Modern Computing* (Cambridge, MA: The MIT Press, 1998): 217.
6. R. X. Gringely, "Be Absolute for Death: Life after Moore's Law," *Communications of the ACM* 44, no. 3 (2001): 94.
7. P.E. Cerruzzi, *A History of Modern Computing* (Cambridge, MA: The MIT Press, 1998): 291.
8. S. Levy, "Back to the Future," *Newsweek*, (21 April 2003).
9. All of these quotes appear in the same article: L. Kappelman, "The Future is Ours," *Communications of the ACM* 44, no. 3 (2001): 46.
10. P. Denning, "Computer Science the Discipline," *Encyclopedia of Computer Science*, ed. E. Reilly, A. Ralston, and D. Hemmendinger (Groves Dictionaries, Inc., 2000).
11. Andrew Tannenbaum. Keynote address at the Technical Symposium of the Special Interest Group on Computer Science Education, San Jose, California, February 1997.
12. P. Denning, D. Comer, D. Gries, M. Mulder, A. Tucker, A. Tuner, and P. Young, "Computing as a Discipline," *Communications of ACM* 32, no. 1(1989): 9–32.

第2章

Biography image: Courtesy of Naval Historical Center.

1. *Webster's New Collegiate Dictionary*, 1977, s.v. "positional notation"
2. Georges Ifrah, *From the Abacus to the Quantum Computer: The Universal History of Computing* (John Wiley & Sons, Inc., 2001): 245.

第3章

Biography image: Courtesy of Bob Bemer.

1. Character set maze from draft article by Bob Bemer.

第4章

Biography Image: © Mary Evans Picture Library/Alamy Images.

1. Written by Chip Weems, adapted from: Nell Dale, Chip Weems, and Mark Headington, *Java and Software Design* (Sudbury, MA: Jones and Bartlett Publishers, Inc., 2001): 242–3.
2. Richard Siegel, "What Is a Nanosecond" (New York).
3. Robert Orr, "Augustus DeMorgan" <http://www.engr.iupui.edu/~orr/webpages/cpt120/mathbios/ademo.htm>

第5章

Biography image: Courtesy of ISU Photo Service.

1. L. Kappelman, "The Future is Ours," *Communications of the ACM* 44, no. 3 (2001): 46.
2. Walter Mossberg, "Personal Technology," *The Wall Street Journal*, (18 January 2001).
3. Alan Perlis, "Epigrams on Programming," *ACM Sigplan Notices* (October, 1981): 7–13.

第6章

Biography Image: © AP Photos.

1. G. Polya, *How to Solve It: A New Aspect of Mathematical Method*, 2d ed., (Princeton, New Jersey: Princeton University Press, 1945).
2. *Houston Junior League CookBook*, The Junior League of Houston, Inc., page 259. E-mail: Cookbook@juniorleaguehouston.org.
3. *Webster's New Collegiate Dictionary*, 1977, s.v. "brainstorming."
4. Grady Booch, "What Is and Isn't Object Oriented Design," *American Programmer* 2, no. 7–8 (Summer 1989).

第7章

Biography image: Courtesy of Los Alamos National Laboratory.

1. Pep/1 through Pep/7 are virtual machines designed by Stanley Warford for his textbook *Computer Systems* (Sudbury, MA: Jones and Bartlett Publishers, Inc., 1999).
2. Stanley Warford, *Computer Systems* (Sudbury, MA: Jones and Bartlett Publishers, Inc., 1999): 146.
3. NIST news release: <http://www.nist.gov/public_affairs/releases/no2-10.htm>

第8章

Biography image: Courtesy of Dianne Driskell, UTCS, The University of Texas at Austin.

1. T. W. Pratt, *Programming Languages: Design and Implementation*, 2d ed., (Englewood Cliffs, NJ: Prentice-Hall, Inc., 1984): 604.
2. Bartleby.com, "Great Books Online" <http://www.bartleby.com>
3. Techtarget.com (2001): <http://WhatIs.techtarget.com>
4. http://www.ultralinguaVB.NET/dictionary/
5. http://www.moneywords.com/glossary
6. K. C. Louden, *Programming Languages: Principles and Practice* (Boston: PWS-KENT Publishing Company, 1993).
7. T. Luce and T. Hankins, *Prolog Minimanual to Accompany Appleby: Programming Languages: Paradigm and Practice* (New York: McGraw-Hill, Inc., 1991).
8. O. Dahl, E. W. Dijkstra, and C. A. R. Hoare, *Structured Programming* (New York: Academic Press, 1972).
9. Stanley Warford, *Computer Systems* (Sudbury, MA: Jones and Bartlett Publishers, Inc., 1999): 222.
10. Records are sometimes called heterogeneous arrays, but this is not common usage.

第9章

Biography image: Courtesy of the Inamori Foundation.

1. Tony Hoare. Adapted from autobiography.
2. <http://www.xprogramming.com/what_is_xp.htm>

第10章

Biography Image: © Christophe Ena/AP Photos.

第11章

Biography Image: Courtesy of Lawrence Lessig, Professor of Law at Stanford Law School.

第12章

Biography image: Louis Fabian Bachrach/Dan Bricklin.

第13章

Biography image: Courtesy of Carnegie Mellon University.

1. PCAI, "Lisp Programming Language," (2001–2002): <http://www.pcai.com/pcai/New_Home_Page/ai_info/pcai_lisp.html>
2. D. Kortenkamp, R. P. Bonasso, and R. Murphy, *Aritificial Intelligence and Mobile Robots* (Menlo Park, California: AAAI Press/The MIT Press, 1998).
3. J. Weizenbaum, *Computer Power and Human Reason* (San Francisco: W. H. Freeman, 1976): 3–4.
4. Mars Now Team and the California Space Institute, 6 October 2001.

5. R. A. Brooks, "A Robust Layered Control System for a Mobile Robot," *IEEE Transactions on Robotics and Automation* 2, no. 1: 14–23.
6. J. H. L. Jones and A. M. Flynn, *Mobile Robots: Inspiration to Implementation* (Wellesley, MA: A K Peters, 1993): 175.

第14章

Biography image: Reproduced by permission of Sun Microsystems.

1. M. Pidd, "An Introduction to Computer Simulation," *Proceedings of the 1994 Winter Simulation Conference.*
2. R. E. Shannon, "Introduction to the Art and Science of Simulation," *Proceedings of the 1998 Winter Simulation Conference.*
3. <http://heim.ifi.uio.no/~Kristen/FORSKNINGSDOK_MAPPE/F_OO_start.html>
4. D. R. Stauffer, N. L. Seaman, T. T. Warner, and A. M. Lario, "Application of an Atmospheric Simulation Model to Diagnose Air-Pollution Transport in the Grand Canyon Region of Arizona," *Chemical Engineering Communications* 121, (1993): 9–25.
5. "Some Operational Forecast Models," *USA Today Weather*, (8 November 2000): <http://www.usatoday.com/weather/wmodlist.htm>
6. D. R. Stauffer, N. L. Seaman, T. T. Warner, and A. M. Lario. "Application of an Atmospheric Simulation Model to Diagnose Air-Pollution Transport in the Grand Canyon Region of Arizona," *Chemical Engineering Communications* 121, (1993): 9–25.
7. Webopedia, s.v. "embedded system," <http://webopedia.internet.com/TERM/e/embedded_system.html>
8. The Ganssle Group. *Microcontroller C Compilers:* <http://www.ganssle.com/articles/acforuc.htm>

第15章

Biography image: Courtesy of Bootstrap Institute.

1. D. Sefton, Newhouse, News Service, *Austin American Statesman* (27 April 2001).
2. M. Softky, "Douglas Engelbart. Computer Visionary Seeks to Boost People's Collective Ability to Confront Complex Problems Coming at a Faster Pace," *The Almanac* (21 February 2001): <http://www.almanacnews.com/thisweek/2001_02_21.cover21.html>

第16章

Biography image: Courtesy of Donna Coveney, MIT.

1. Jessica Mintz, "When Bloggers Make News," *The Wall Street Journal,* January 21, 2005, p. B1.
2. Editorial, "The Apple Case Isn't Just a Blow to Bloggers," *Business Week,* March 28, 2005, p. 128.

3. St. Thomas Aquinas, *Summa Theologiae,* (New York: Benziger Bros, 1947), IIa-IIa, 109-110.

4. Jessica Mintz, "When Bloggers Make News," *The Wall Street Journal,* January 21, 2005, p. B4.

第17章

Biography image: By courtesy of the National Portrait Gallery, London.

1. H. M. Walker, *The Limits of Computing* (Sudbury, MA: Jones and Bartlett Publishers, 1994). This fable and many of the ideas in this chapter come from Dr. Walker's thought-provoking little book. Thank you, Henry.

2. *Software Engineering Note* 11, no. 1 (January 1986): 14.

3. John Markoff, "Circuit Flaw Causes Pentium Chip to Miscalculate, Intel Admits," *The New York Times* (24 November 1994), c. 1991, N.Y. Times News Service.

4. N. G. Leveson, "Software Engineering: Stretching the Limits of Complexity," *Communications of the ACM* 40, no. 2 (February 1997): 129.

5. D. Bell, I. Morrey, and J. Pugh, *Software Engineering, A Programming Approach*, 2 ed. (Prentice Hall, 1992).

6. T. Huckle, *Collection of Software Bugs:* <http://wwwzenger.informatik.tu-muenchen.de/persons/huckle/bugse.html#general>

7. Dennis Beeson, Manager, Naval Air Warfare Center, Weapons Division, F–18 Software Development Team.

8. D. Gries, "Queen's Awards Go to Oxford University Computing and INMOS," *Computing Research News* 2, no. 3 (July 1990): 11.

9. "Out in the open," *The Economist*, (14–20 April 2001).

10. "Out in the open," *The Economist*, (14–20 April 2001).

11. Edsger Dijkstra, "On the Cruelty of Really Teaching Computing Science," *Communications of the ACM* 32, no. 12 (December 1989): 1402.

12. "Ghost in the Machine," *TIME—The Weekly Newsmagazine*, (29 January 1990): 59.

13. T. Huckle, *Collection of Software Bugs:* <http://wwwzenger.informatik.tu-muenchen.de/persons/huckle/bugse.html#general>

14. N. G. Leveson and C. S. Turner, "An Investigation of the Therac–25 Accidents," *IEEE Computer* 26, no. 7 (July 1993): 18–41.

15. Douglas Arnold, *The Patriot Missile Failure:* <http://www.math.psu.edu/dna/disasters/patriot.html>

16. United States General Accounting Office Information Management and Technology Division, B–247094 (February 4, 1992).

17. J. Fox, *Software and Its Development* (Englewood Cliffs, New Jersey: Prentice-Hall, 1982): 187–188.

18. Douglas Isbell and Don Savage, *Mars Polar Lander* (1999): <http://mars.jpl.nasa.gov/msp98/news/mco991110.html>

19. Paul Gray, "Computer Scientist Alan Turing" <http://www.time.com/time/time100/scientist/profile/turing.html>

20. Paul Gray, "Computer scientist Alan Turing," *Time Magazine*, (29 March 1999).

［Ｇｅｎｅｒａｌ Ｉｎｆｏｒｍａｔｉｏｎ］

书名：计算机科学概论

作者：（美）戴尔，（美）刘易斯著，张欣等译

页数：３７５

出版社：机械工业出版社

出版日期：２００９．０２

简介：本书由两位知名的计算机科学教育家编写，全面而细致地介绍了计算机科学的各个方面。书中，计算系统的每个分层都以剖析，从住处层开始，历经硬件层、程序设计层、操作系统层、应用程序层的通信层，最后讨论了计算的限制。此外，正文中穿插了大量的人物传记、历史注释、道德问题和最新的技术发展信息，有助于你进一步了解计算机科学。每章后面都附带有大量的练习，可以帮助你即时重温并掌握这一章所述的内容。　　　　本书是计算和计算机科学引论课程的理想教材，对于想要了解计算机科学概况的非专业人员，本书也是一个很好的选择。

ＳＳ号：１２７３８９５９

ＤＸ号：０００００６６７５８０８

ｈｔｔｐ：／／ｂｏｏｋ．ｄｕｘｉｕ．ｃｏｍ／ｂｏｏｋＤｅｔａｉｌ．ｊｓｐ？ｄｘＮｕｍｂｅｒ＝０００００６６７５８０８＆ｄ＝８Ｂ８４Ｂ７６Ｅ２６８９ＤＢＤ７２Ａ００Ａ４Ｃ１４Ｂ１７０９４２＆ｆｅｎｌｅｉ＝１８１７０４＆ｓｗ＝％ＢＣ％Ｃ６％ＣＢ％Ｅ３％ＢＢ％ＦＡ％ＢＦ％Ｃ６％Ｄ１％Ａ７％Ｂ８％Ｃ５％Ｃ２％ＤＢ

封面
书名
版权
前言
目录

第一部分　基础篇第１章　全景图
１．１　计算系统
１．１．１　计算系统的分层
１．１．２　抽象
１．２　计算的历史
１．２．１　计算硬件的简史
１．２．２　计算软件的简史
１．２．３　预言
１．３　计算工具和计算学科
小结
道德问题：数字化分裂
练习
思考题
第二部分　信息层第２章　二进制数值和记数系统
２．１　数字和计算
２．２　位置记数法
２．２．１　二进制、八进制和十六进制
２．２．２　其他记数系统中的运算
２．２．３　以２的幂为基数的记数系统
２．２．４　把十进制数转换成其他数制的数
２．２．５　二进制数值和计算机
小结
道德问题：计算机和国家安全
练习
思考题
第３章　数据表示法
３．１　数据和计算机
３．１．１　模拟数据和数字数据
３．１．２　二进制表示法
３．２　数字数据的表示法
３．２．１　负数表示法
３．２．２　实数表示法
３．３　文本表示法
３．３．１　ＡＳＣＩＩ字符集
３．３．２　Ｕｎｉｃｏｄｅ字符集
３．３．３　文本压缩
３．４　音频信息表示法
３．４．１　音频格式
３．４．２　ＭＰ３音频格式
３．５　图像和图形的表示法
３．５．１　颜色表示法
３．５．２　数字化图像和图形
３．５．３　图形的矢量表示法
３．６　视频表示法
小结
道德问题：ＭＧＭ　Ｓｔｕｄｉｏｓ公司和Ｇｒｏｋｓｔｅｒ有限公司
练习
思考题
第三部分　硬件层第４章　门和电路
４．１　计算机和电学
４．２　门
４．２．１　非门
４．２．２　与门
４．２．３　或门
４．２．４　异或门
４．２．５　与非门和或非门
４．２．６　门处理回顾
４．２．７　具有更多输入的门
４．３　门的构造
４．４　电路
４．４．１　组合电路
４．４．２　加法器
４．４．３　多路复用器
４．５　存储器电路
４．６　集成电路
４．７　ＣＰＵ芯片
小结
道德问题：电子邮件隐私权
练习
思考题
第５章　计算部件
５．１　独立的计算机部件
５．２　存储程序的概念
５．２．１　冯·诺伊曼体系结构

５．２．２　读取－执行周期
５．２．３　ＲＡＭ和ＲＯＭ
５．２．４　二级存储设备
５．２．５　触摸屏
５．３　非冯·诺伊曼体系结构
小结
道德问题：生物信息学研究和ｄｅＣＯＤＥ　Ｇｅｎｅｔｉｃｓ公司的案例
练习
思考题
第四部分　程序设计层第６章　问题求解和算法设计
６．１　问题求解
６．１．１　如何解决问题
６．１．２　应用Ｐｏｌｙａ的问题求解策略
６．２　算法
６．２．１　计算机问题求解
６．２．２　执行算法
６．２．３　开发算法
６．３　伪代码
６．３．１　执行一个伪代码算法
６．３．２　伪代码的功能
６．３．３　伪代码示例
６．４　自顶向下设计方法
６．４．１　一个通用的实例
６．４．２　一个计算机实例
６．４．３　方法总结
６．４．４　测试算法
６．５　面向对象方法
６．５．１　面向对象
６．５．２　设计方法
６．５．３　一个通用的实例
６．５．４　一个计算机实例
６．６　几个重要思想
６．６．１　信息隐蔽
６．６．２　抽象
６．６．３　事物命名
６．６．４　程序设计语言
６．６．５　测试
小结
道德问题：计算机专业人员许可
练习
思考题
第７章　低级程序设计语言
７．１　计算机操作
７．２　抽象的分层
７．３　机器语言
７．４　一个程序实例
７．４．１　问题和算法
７．４．２　程序
７．５　汇编语言
７．５．１　Ｐｅｐ／７汇编语言
７．５．２　伪代码操作
７．５．３　“Ｈｅｌｌｏ”程序的汇编语言版本
７．５．４　一个新程序
７．５．５　具有分支的程序
７．５．６　具有循环的程序
７．６　其他重要思想
７．６．１　抽象
７．６．２　测试
７．６．３　测试计划实现
小结
道德问题：软件盗版和版权
练习
思考题
第８章　高级程序设计语言
８．１　翻译过程
８．１．１　编译器
８．１．２　解释器
８．２　程序设计语言的范型
８．３　命令式语言的功能性
８．３．１　布尔表达式
８．３．２　强类型化
８．３．３　输入／输出结构
８．３．４　控制结构
８．３．５　复合数据类型
８．４　面向对象语言的功能性
８．４．１　封装
８．４．２　继承
８．４．３　多态性
小结

道德问题：开源软件的发展
练习
思考题
第９章　抽象数据类型和算法
９．１　抽象数据类型
９．２　实现
９．２．１　基于数组的实现
９．２．２　链式实现
９．３　列表
９．３．１　列表的基本操作
９．３．２　其他列表操作
９．４　排序
９．４．１　选择排序
９．４．２　冒泡排序
９．４．３　快速排序
９．５　二分检索法
９．６　栈和队列
９．６．１　栈
９．６．２　队列
９．６．３　实现
９．７　树
９．７．１　二叉树
９．７．２　二叉检索树
９．７．３　其他操作
９．７．４　图
９．８　程序设计库
小结
道德问题：使用计算机的恶作剧和欺诈行为
练习
思考题
第五部分　操作系统层第１０章　操作系统
１０．１　操作系统的角色
１０．１．１　内存、进程和ＣＰＵ管理
１０．１．２　批处理
１０．１．３　分时操作
１０．１．４　其他ＯＳ要素
１０．２　内存管理
１０．２．１　单块内存管理
１０．２．２　分区内存管理
１０．２．３　页式内存管理
１０．３　进程管理
１０．３．１　进程状态
１０．３．２　进程控制块
１０．４　ＣＰＵ调度
１０．４．１　先到先服务
１０．４．２　最短作业优先
１０．４．３　循环调度法
小结
道德问题：数字版权管理和关于Ｓｏｎｙ公司的根目录案件的争论
练习
思考题
第１１章　文件系统和目录
１１．１　文件系统
１１．１．１　文本文件和二进制文件
１１．１．２　文件类型
１１．１．３　文件操作
１１．１．４　文件访问
１１．１．５　文件保护
１１．２　目录
１１．２．１　目录树
１１．２．２　路径名
１１．３　磁盘调度
１１．３．１　先到先服务磁盘调度法
１１．３．２　最短寻道时间优先磁盘调度法
１１．３．３　ＳＣＡＮ磁盘调度法
小结
道德问题：垃圾邮件
练习
思考题
第六部分　应用程序层第１２章　信息系统
１２．１　信息管理
１２．２　电子制表软件
１２．２．１　电子数据表公式
１２．２．２　循环引用
１２．２．３　电子数据表分析
１２．３　数据库管理系统
１２．３．１　关系模型
１２．３．２　关系
１２．３．３　结构化查询语言

１２．３．４　数据库设计
１２．４　信息安全
１２．４．１　机密性、完整性和可用性
１２．４．２　密码学
小结
道德问题：加密
练习
思考题
第１３章　人工智能
１３．１　思维机
１３．１．１　图灵测试
１３．１．２　ＡＩ问题的各个方面
１３．２　知识表示
１３．２．１　语义网
１３．２．２　检索树
１３．３　专家系统
１３．４　神经网络
１３．４．１　生物神经网络
１３．４．２　人工神经网络
１３．５　自然语言处理
１３．５．１　语音合成
１３．５．２　语音识别
１３．５．３　自然语言理解
１３．６　机器人学
１３．６．１　感知－规划－执行范型
１３．６．２　包孕体系结构
１３．６．３　物理部件
小结
道德问题：ＨＩＰＡＡ（健康保险携带和责任法案）
练习
思考题
第１４章　模拟、图形学和其他应用程序
１４．１　什么是模拟
１４．１．１　复杂系统
１４．１．２　模型
１４．１．３　构造模型
１４．１．４　排队系统
１４．１．５　气象模型
１４．１．６　其他模型
１４．１．７　必要的计算能力
１４．２　计算机图形学
１４．２．１　光的工作原理
１４．２．２　物体形状
１４．２．３　光模拟
１４．２．４　复杂对象的建模
１４．２．５　让物体动起来
１４．３　嵌入式系统
１４．４　电子商务
１４．５　计算机安全
１４．５．１　恶意代码
１４．５．２　安全攻击
小结
道德问题：入侵大学的计算机系统，查询录取程序中某人的录取状态
练习
思考题
第七部分　通信层第１５章　网络
１５．１　连网
１５．１．１　网络的类型
１５．１．２　Ｉｎｔｅｒｎｅｔ连接
１５．１．３　包交换
１５．２　开放式系统和协议
１５．２．１　开放式系统
１５．２．２　网络协议
１５．２．３　ＴＣＰ／ＩＰ
１５．２．４　高层协议
１５．２．５　ＭＩＭＥ类型
１５．２．６　防火墙
１５．３　网络地址
小结
道德问题：无所不在的计算
练习
思考题
第１６章　万维网
１６．１　Ｗｅｂ简介
１６．１．１　搜索引擎
１６．１．２　即时消息
１６．１．３　博客
１６．１．４　ｃｏｏｋｉｅ
１６．２　ＨＴＭＬ

１６．２．１　　基本的ＨＴＭＬ格式
１６．２．２　　图像和链接
１６．３　　交互式Ｗｅｂ页
１６．３．１　　Ｊａｖａ小程序
１６．３．２　　Ｊａｖａ服务器页
１６．４　　ＸＭＬ
小结
道德问题：写博客
练习
思考题
第八部分　　总结
第１７章　　计算的限制
１７．１　　硬件
１７．１．１　　算术运算的限制
１７．１．２　　部件的限制
１７．１．３　　通信的限制
１７．２　　软件
１７．２．１　　软件的复杂度
１７．２．２　　当前提高软件质量的方法
１７．２．３　　臭名昭著的软件错误
１７．３　　问题
１７．３．１　　算法比较
１７．３．２　　图灵机
１７．３．３　　停机问题
１７．３．４　　算法分类
小结
道德问题：深度链接
练习
思考题
参考文献